MASTER EN SCIENCES INFORMATIQUES

La créativité graphique de la personne ayant une déficience intellectuelle : un logiciel d'aide à l'élaboration de Bandes Dessinées

Pequet, Benoît; Tillieux, Marc

Award date:
1992

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Faculté d'Informatique
Rue Grandgagnage, 21, 5000 Namur

La créativité graphique de la personne ayant une déficience intellectuelle : un logiciel d'aide à l'élaboration de Bandes Dessinées

Benoît PEQUET **Marc TILLIEUX**

Mémoire présenté en vue de l'obtention du titre de
Licencié et Maître en Informatique

Promoteur : Madame le Professeur Monique NOIRHOMME-FRAITURE
(Institut d'Informatique)
Co-Promoteur : Monsieur le Professeur Michel MERCIER
(Département de psychologie de la Faculté de Médecine)

JUIN 1992

Remerciements

C'est pour nous un plaisir d'exprimer notre plus vive reconnaissance à toutes les personnes qui ont collaboré, sous divers aspects, à la réalisation de ce mémoire.

Nous remercions plus spécialement:

Madame le Professeur Monique Noirhomme-Fraiture, promoteur du présent mémoire, pour l'aide qu'elle nous a procuré tout au long de l'élaboration de ce travail, ainsi que pour l'intérêt qu'elle y a accordé et les judicieux conseils qu'elle nous a prodigués.

Monsieur le Professeur Michel Mercier, co-promoteur du mémoire, pour l'accueil qu'il nous a réservé au sein du département de psychologie (dont il est le directeur) des facultés de Médecine, et pour toutes les informations fort instructives et importantes qu'il nous a données.

Madame Odette Witdouck, assistante dans le département de psychologie, animateur au centre PSINHA de ce même département, qui nous a éclairé dans des matières relativement méconnues et qui a assuré un suivi constant en nous faisant part, entre autre, de son expérience dans le monde de l'informatique pour des personnes handicapées mentales.

Monsieur le Professeur Baudouin Lecharlier, pour l'aide précieuse fournie quant à la bonne compréhension de certains concepts orientés objet.

Monsieur Bruce Kent, qui nous a chaleureusement accueilli à Dartington et guidé lors de notre stage en Angleterre.

Monsieur Luc Boulangé, à qui l'on doit l'idée de départ du logiciel et qui nous a fourni des renseignements concernant les spécifications du programme.

Nous sommes aussi profondémment reconnaissants envers nos familles, amis et collègues, pour leur soutien continu, et plus particulièrement envers nos parents qui nous ont permis d'entreprendre ces années d'études et d'aboutir ainsi à l'élaboration de ce mémoire.

Résumé

Nous avons conçu, dans le cadre de ce mémoire, un logiciel d'aide à la création de Bandes Dessinées pour des personnes ayant une déficience mentale. Toutes les étapes, de l'énoncé initial à la programmation même du logiciel, sont présentées. Toutefois, étant donné que nous nous sommes référencés à la théorie orientée objet pour la conception de ce logiciel, nous mettons essentiellement l'accent sur cette théorie et sur la méthode mise en oeuvre. Nous insistons également sur l'élaboration de l'interface homme/machine qui, dans notre contexte, est particulièrement importante.

Abstract

We have realised, in the scope of this thesis, a software for helping mentally handicaped people in the creation of comic strips. All the steps, from the initial terms to the programmation itself, is presented. Nevertheless, provided that we used object oriented theory for the design of the software, we mainly focus our attention on this theory and on the method followed. We also put the stress on the human/machine interface which, in our context, is mostly important.

Table des matières

Introduction

Les domaines concernés par l'informatique au service de la personne ayant une déficience mentale peuvent être nombreux et concerner des sujets assez divers. Ainsi en est-il des domaines de l'autonomie et de connaissance du corps qui se sont enrichis, ces derniers temps, de nouveaux logiciels. Par contre, il existe d'autres domaines pour lesquels peu d'efforts ont été, jusqu'à ce jour, consentis. Ainsi, le thème de la créativité par informatique pour cette personne ne fut que trop rarement abordé. Il n'existe, en effet, pour l'heure, pratiquement aucune recherche ni données scientifiques dans ce domaine.

Il est cependant de plus en plus admis que la technologie actuelle permet d'augmenter le degré d'ouverture de la personne mentalement déficiente avec son entourage en lui fournissant des moyens de communication et d'expression nouveaux. Ces techniques, quoique de plus en plus élaborées et sophistiquées, ne leur sont pas pour autant difficilement accessibles. Elles ne nécessitent en effet qu'un certain apprentissage à condition, bien évidemment, qu'elles aient été préparées à leur intention.

Le sujet de ce mémoire est de mettre des outils informatiques à la disposition d'une partie de la population ayant un handicap mental et ce, dans le domaine de la créativité graphique. Ces outils dont la complexité (de programmation) augmente avec les capacités techniques, doivent notamment, pour leur rester accessibles, disposer d'une interface homme/machine adaptée à leurs besoins. Nous nous intéressons ici plus particulièrement au domaine de la création de bandes dessinées. Nous avons, en effet, étudié la réalisation d'un logiciel, appelé "Logiciel BD", permettant à des utilisateurs handicapés mentaux, modérés ou légers, de créer leurs propres bandes dessinées.

L'idée de départ de ce logiciel provient d'animateurs d'ateliers créatifs spécialisés dans le graphisme chez les personnes mentalement déficientes.

Nous avons alors été en contact avec des psychologues et de tels animateurs tout au long de l'élaboration du travail, afin de guider notre démarche auprès de la population cible du logiciel. Le premier chapitre de ce mémoire commence ainsi par situer notre travail dans son environnement psychologique. C'est aussi là que nous dégageons les fondements de notre logiciel dans le domaine de la créativité. Nous avons donc tenté de présenter la notion complexe de créativité graphique, plus particulièrement dans l'utilisation de l'informatique. Nous y avons également ajouté une description de la Bande Dessinée et du rôle qu'elle peut avoir auprès de ces personnes.

Il nous a ensuite semblé important de présenter le contexte dans lequel est né le projet pour arriver à l'élaboration d'un énoncé définitif de "Logiciel BD". Chaque élément qui constitue ce logiciel y est décrit et justifié par rapport aux expériences que nous avons eues dans des ateliers créatifs ou en fonction de recommandations de psychologues et d'animateurs. Cette phase de définition du logiciel représente un travail de plusieurs mois et est décrite dans le second chapitre de ce mémoire.

Le troisième chapitre met en évidence le travail de modélisation des informations et des traitements du logiciel. Cette phase de modélisation vient à la suite de l'élaboration de l'énoncé. Nous avons estimé qu'elle était nécessaire car elle spécifie de façon formelle tous les traitements et informations manipulés par le logiciel et qui peuvent être dégagés de l'énoncé.

Le quatrième chapitre présente les notions de la théorie orientée objet qui propose une approche de conception de logiciels que nous avons décidé de mettre en oeuvre pour "Logiciel BD". C'est dans ce chapitre que nous présentons les buts qu'une telle méthode poursuit et la manière de modéliser les traitements et les informations qu'elle propose. Il s'agit d'abord d'une présentation purement théorique des concepts de base de l'approche orientée objet. Viennent ensuite les mécanismes techniques qui sont proposés pour mettre en oeuvre cette théorie. Nous pensons qu'il est approprié de séparer, dans ce chapitre, l'approche théorique d'une approche plus technique. En effet, les mécanismes orientés objet décrits ne sont qu'une manière de mettre en oeuvre, de façon souvent incomplète, l'idée qui se trouve derrière la théorie orientée objet.

Dans le cinquième chapitre, nous justifions les raisons qui nous ont poussés à suivre l'approche orientée objet pour la conception de "Logiciel BD", tout en présentant ce qu'elle nous a réellement apporté comme avantages et inconvénents par rapport à une autre. Cependant, la partie du projet "Logiciel BD" que nous avons implémentée l'a été dans un langage non-orienté objet, à savoir le langage C. C'est pour cette raison que nous présentons des exemples montrant comment il est possible d'implémenter la plupart des mécanismes, présentés au chapitre 4, dans le langage non orienté objet qu'est le langage C.

Comme nous manquions de pratique en ce qui concerne la conception orientée objet de logiciels, nous avons jugé utile de baser notre démarche sur une méthodologie particulière. C'est le but du sixième chapitre que de présenter les différentes étapes de cette méthodologie ainsi que la manière dont nous l'avons suivie.

Dans le septième chapitre, nous étudions la conception de l'interface homme/machine du logiciel. Nous la présentons dans un chapitre à part dû à l'importance que nous lui avons accordée dans la réalisation de "Logiciel BD". En effet, la conception de l'interface revêt une importance considérable lorsqu'il s'agit de réaliser un logiciel pour une population particulière. Elle est d'autant plus importante dans le cadre de notre mémoire étant donné la haute interactivité de "Logiciel BD". Nous avons donc essayé de réaliser une interface qui soit la plus "proche" possible de la population cible. Pour ce faire, nous nous sommes basés sur des principes théoriques généraux d'un part, et particuliers aux personnes mentalement déficientes d'autre part. Dans ce chapitre, nous présentons ces différents principes ainsi que la manière dont nous les avons mis en oeuvre pour l'interface du logiciel.

Finalement, le "Logiciel BD" étant un logiciel permettant de travailler dans le domaine du graphisme au moyen de l'ordinateur, nous présentons, dans le huitième chapitre, quelques notions de base de manipulations et de stockage informatiques d'images graphiques. Nous y expliquons, au moyen de ces notions de base, les situations particulières auxquelles nous avons été confrontés.

Chapitre I

Aspects psychologiques de "Logiciel BD"

I.1. INTRODUCTION

Nous présentons, par l'intermédiaire de ce premier chapitre, les différents aspects psychologiques de notre logiciel de création de bandes dessinées, le **"Logiciel BD".** Ce logiciel étant destiné aux personnes ayant un handicap mental (léger ou modéré), nous décrivons dans un premier temps le concept de **handicap mental.** Nous pensons ainsi sensibiliser le lecteur à la complexité du projet considéré du point de vue psychologique. Dans un second temps, l'intérêt primordial de ce logiciel étant de permettre à cette population d'exprimer sa créativité au travers du graphisme, il nous paraît essentiel de présenter la notion de **créativité** auprès de ces personnes afin de voir comment elle peut se manifester, par la suite, au travers de l'**art graphique.** L'outil d'expression de la créativité étant, dans le projet qui nous préoccupe, la **micro-informatique,** nous décrivons ensuite le rôle qu'elle peut jouer dans l'expression de cette créativité, toujours auprès de cette population. Le but du logiciel étant de produire des pages de bandes dessinées, nous

définissons le terme **"Bandes Dessinées"** ainsi que les caractéristiques de ce moyen d'expression. Finalement, nous terminons ce chapitre par une présentation des intérêts suscités par la création de bandes dessinées chez les personnes mentalement déficientes.

I.2. LE HANDICAP MENTAL

Dans ce point, nous tentons de donner au lecteur non averti un aperçu de la définition du handicap mental et de ses composants ainsi que d'effectuer un survol rapide de l'étiologie (l'étude des causes) et des théories existantes quant au concept de handicap mental. Nous terminons par une présentation des méthodes médicales et psychologiques d'intervention.

I.2.1. Définitions et dimensions du handicap mental

Nous présentons deux approches "opposées" de la définition du handicap mental. Ensuite, nous décrivons brièvement les différentes dimensions concernant cette définition. Nous les complétons finalement en introduisant, très succinctement, la notion de validation écologique des conduites.

I.2.1.1. Définitions du handicap mental

La première approche, défendue par Jean-Luc Lambert, définit le handicap mental comme étant "un problème pratique, multivarié, dont la compréhension exige l'intégration de plusieurs dimensions" [LAMBE,86]. Il existe bien sûr d'autres définitions parmi lesquelles celle proposée, en 1973, par l'Association Américaine de la Déficience Mentale, et plus précisément par Grossman, qui se réfère, pour définir le handicap mental, à 3 critères principaux: un fonctionnement intellectuel général significativement inférieur à la moyenne, un déficit du comportement adaptatif et le fait que la déficience mentale se manifeste durant la période développementale.

Cependant, si la définition de Grossman fut tout un temps très référencée, il nous est apparu que celle de Lambert, au vu de son contenu et des dimensions[1] qu'elle intègre, représentait de façon plus adéquate les connaissances actuelles relatives au handicap mental. Nous constatons donc que le point de vue de Jean-Luc Lambert respecte une vision pratique et complexe du sujet. Cette vision résulte de l'hétérogénéité -due à la différence des degrés du handicap- des populations concernées par le handicap mental et de la diversité des opinions qui peuvent être formulées par différents observateurs qu'ils soient médecins, psychologues, sociologues, éducateurs ou encore juristes.

La seconde approche, beaucoup plus quantifiable, prônée entre autres par Zigler, consiste à se baser sur le concept d'intelligence et d'utiliser uniquement la notion du Quotient Intellectuel (**Q.I.**) pour définir le handicap mental. Cependant, cette perception du problème s'avère être trop réductionniste, même si l'auteur se défend en avançant l'argument suivant: "L'utilisation du QI au centre de la définition du handicap mental dérive du fait que les sciences comportementales n'ont pas actuellement de meilleurs instruments à proposer pour mesurer le fonctionnement intellectuel" [ZIGLE,84]. En effet, l'argument le plus défavorable à cette vision du handicap est que le concept d'intelligence n'est pas, en lui-même, suffisamment compris et déterminé au niveau scientifique. Comment peut-on alors appréhender la réalité du problème du handicap mental, surtout si l'on désire se baser sur le QI?

Nous constatons donc qu'il existe au moins deux approches différentes concernant la définiton du handicap mental. De plus, bien que chacune d'entre-elles référence la notion de QI, elles l'utilisent à des degrés d'incorporation divers. D'où il existe des différences relatives entre les différentes approches quant à la répartition et l'évaluation de la population concernée.

La notion de QI a été introduite en 1916 par Terman et permet de mesurer la vitesse du développement intellectuel ainsi que de grouper les individus sous forme de classes. Ce concept représente

---

[1] cfr. infra

en fait le rapport entre l'âge mental[2] (mesuré par un test standardisé) et l'âge chronologique[3]. L'AM définit le niveau intellectuel de l'individu ou, autrement dit, ce qu'il est capable de faire, et l'AC donne le temps mis par la personne pour atteindre ce niveau. Quant aux différents niveaux du QI qui permettent d'établir un diagnostic, une personne sera reconnue comme présentant une déficience mentale si elle obtient, lors d'un ou de plusieurs tests standardisés d'intelligence générale, "un QI inférieur ou égal à 70" [GROSS,83]. En dessous de ce seuil, 4 niveaux de handicap mental peuvent être mis en évidence: "léger (QI de 50-55 à environ 70), modéré (QI de 35-40 à 50-55), sévère (QI de 20-25 à 35-40) et profond (en dessous de 20 ou 25)" [IONES,87]. Afin de rendre plus souple l'utilisation de ce critère, nous constatons que "la limite des différents niveaux est située sur une bande de cinq points" [GROSS,83].

Il est cependant "insensé", selon Weizenbaum, de vouloir quantifier l'intelligence: "L'intelligence par elle-même est un concept sans signification. Elle exige, pour devenir significative, un cadre de référence (c'est-à-dire un acquis social et culturel avec ses domaines caractéristiques de pensée et d'action) ainsi que la spécification d'un domaine de pensée et d'action" [WEIZE,81]. Non pas que le test du QI soit faux, mais qu'il est tout simplement incomplet et ce, pour deux raisons. La première raison est que la créativité humaine est fonction de l'interaction entre la faculté de forger et de saisir des concepts et d'autres formes de pensée (telles que l'intuition, la sagesse,...). La seconde est que l'intelligence n'est pas un phénomène linéairement mesurable et qu'elle n'est pas indépendante de tout cadre de référence. Néanmoins, "l'idée que l'intelligence est mesurable sur une échelle absolue, et donc que les intelligences sont comparables, a profondémment pénétré le sens commun" [WEIZE,81].

I.2.1.2. Dimensions du handicap mental

---

[2] noté AM
[3] noté AC

Les dimensions qui, selon Jean-Luc Lambert, peuvent être intégrées dans la définition relative au handicap mental sont les dimensions intellectuelle, personnelle, sociale et culturelle.

* *La dimension intellectuelle*

Le niveau intellectuel de tout individu a été, et est encore de nos jours, bien souvent défini par les limites du QI. Cependant, cette vision réduite de l'intelligence - et par là même de la problématique - ne reflète pas exactement la réalité. Afin d'évaluer de façon adéquate l'intelligence d'une personne, la réalité et l'environnement socio-culturels de l'individu doivent être pris en compte. Alors seulement, la dimension intellectuelle peut être intégrée à la compréhension de la notion de handicap mental.

* *La dimension personnelle*

Nous constatons très fréquemment que le comportement d'une personne est également défini par des facteurs motivationnels et émotionnels comme, par exemple, la crainte permanente de l'échec. Sinon, comment pourrait-on expliquer, même si cette justification n'est que partielle (d'autres dimensions doivent intervenir pour la compléter), que deux individus qui ont le même âge, le même niveau intellectuel ou le même QI, aient un comportement quotidien différent?

* *La dimension sociale*

Chez une personne handicapée mentale, nous détectons non seulement des troubles intellectuels, mais également des déficits quant à son adaptation sociale. Cette adaptation, ou compétence sociale, est "le degré avec lequel l'individu rencontre les normes de maturation, d'apprentissage, d'indépendance personnelle et de responsabilité sociale attendues pour son âge et en référence à son appartenance culturelle" [GROSS,83]. La notion d'inadaptation sociale peut ne prendre sa signification qu'à certaine(s) période(s). Par exemple, un individu peut être considéré, durant sa scolarité, comme une personne handicapée mentale. Ceci permettra de le placer dans un enseignement spécialisé. Après cette période, l'individu peut très bien parvenir à un niveau d'adaptation

socioprofessionnelle adapté à son niveau intellectuel, et ne plus être alors considéré comme une personne mentalement déficiente.

* *La dimension culturelle*

L'intégration de la dimension culturelle dans la définition du handicap mental a contribué à modifier la perception identifiant une personne handicapée mentale à son déficit intellectuel, à souligner l'importance des concepts d'intégration et de normalisation, ainsi qu'à rendre réalisable l'évaluation des individus dans leur langue maternelle. La prise en compte de cette dimension a donc permis, d'un côté, d'élargir le champ de perception de la notion du handicap mental mais, d'un autre côté, d'en rendre plus complexes la compréhension et la définition.

I.2.1.3. Validation écologique des conduites

L'introduction de la notion de validation écologique des conduites (instinctives, cognitives, émotionnelles, apprises, représentatives,...) a offert, dans le domaine du handicap mental, selon J-L. Lambert, de nouvelles perspectives au niveau des systèmes d'évaluation et des moyens d'intervention.

Du point de vue des sciences du comportement, "l'écologie se réfère à un courant théorique particulier dont l'objectif principal est d'élaborer un ensemble cohérent de connaissances portant sur les relations entre les conduites et l'environnement, en tant qu'éléments interdépendants faisant partie d'un système unitaire et dynamique" [LAMBE,86].

Concrètement, un domaine écologique -du handicap mental- fait référence à trois niveaux différents d'analyse, à savoir les niveaux descriptif, méthodologique et théorique. Il faut entendre par écologique, pour le niveau descriptif, l'observation des conduites des individus mentalement handicapés évoluant dans leurs milieux de vie. En ce qui concerne le deuxième niveau, ce courant met en évidence la prise en compte du contexte social dans les processus de

prise de décision des moyens d'intervention adéquats aux personnes ayant une déficience mentale. Quant au niveau théorique, la notion de validation écologique a trait directement à la définition du concept du handicap mental.

I.2.1.4. Conclusion

Nous nous rendons compte que la définition du handicap mental et, par conséquent, l'évaluation des individus sont des problèmes beaucoup moins aisé que ce que le sens commun nous donne à penser. Un individu est un ensemble complexe qui comporte plusieurs aspects ou variables en relation dynamique.

Afin de pouvoir "bien" évaluer le problème concerné, il faut être capable de déterminer convenablement tout ce qui y a trait, et principalement les différentes dimensions à prendre en compte. De fait, "la compréhension du handicap mental procède d'un système complexe d'interactions impliquant deux composantes: l'individu, donnée biologique irréductible, et son environnement, système dont le fonctionnement demande la prise en considération de trois dimensions interdépendantes: personnelle, dans ses composantes intellectuelles et motivationnelles, sociale, dans la perspective de l'adaptation de l'individu à son milieu, et culturelle"[4] [LAMBE,86].

La diversité des approches vient du fait que chaque psychologue ne considère que l'une ou l'autre dimension du problème. L'utilisation stricte du Q.I. réduit à une échelle unidimensionnel un aspect beaucoup plus complexe qui contient plus de variables. Le problème vient du fait que l'être humain n'est vraiment capable de comparer efficacement que des objets univariés. C'est ainsi que nous avons souvent tendance à réduire les objets de notre univers à une seule mesure, quand nous ne raisonnons pas de façon dichotomique: vrai/faux, bon/mauvais...

---

[4] Cette dimension personnelle regroupe les dimensions personnelle et intellectuelle présentées précédemment.

I.2.2. Etiologie

L'interaction entre l'héritage biologique d'un être humain et l'environnement dans lequel il vit, détermine son comportement. Cependant, certains dérèglements peuvent perturber ce développement humain et engendrer un handicap mental chez l'individu.

Les causes de ce dysfonctionnement sont, selon Jean-Luc Lambert, de deux types: soit les processus organiques pathologiques, soit des interactions entre des influences génétiques et environnementales. Le premier type est (presque) directement responsable de la déficience mentale, et le deuxième est souvent suffisant pour provoquer un dérèglement intellectuel (mais rarement un trouble organique). Alors que les causes d'ordre pathologique engendrent des formes de handicap modéré, sévère et profond, les autres ne conduisent qu'à une déficience mentale légère.

Comme facteurs provoquant des troubles organiques chez une personne, nous distinguons, toujours selon J-L. Lambert, les facteurs génétiques des facteurs environnementaux. Il existe, en ce qui concerne les facteurs génétiques, les syndrômes dûs à des anomalies des chromosomes (tels le syndrôme de Down -le Mongolisme-, le syndrôme de Turner -une petite taille-,...), les syndrômes dûs à des perturbations des gênes dominants et les syndrômes dûs à des anomalies des gênes récessifs. Quant aux facteurs environnementaux, nous trouvons des causes prénatales (rubéole, drogues,...), des causes périnatales et des causes postnatales (la méningite tuberculeuse,...).

I.2.3. Théories du handicap mental

Les théories relatives au handicap mental, issues directement de l'expérimentation, sont, d'après J-L. Lambert, au nombre de trois. L'une d'elles est la théorie déficitaire. Elle compare, quantitativement et qualitativement, les personnes dites normales aux individus handicapés mentaux. Sa formulation est la suivante: "Les arriérés mentaux présentent un comportement retardé lorqu'ils sont comparés à des sujets normaux d'âge chronologique identique. Les

différences entre les arriérés et les normaux étant comportementales." [LAMBE,78].

Une autre théorie, prônant une différence principalement quantitative entre la personne ayant un handicap mental et la personne dite normale, est la théorie développementale. Elle affirme que "le développement cognitif de l'arriéré mental est caractérisé par une progression plus lente, mais identique à celle de la personne normale, c'est-à-dire en progressant selon la même séquence de niveaux cognitifs" [LAMBE,78].

La troisième et dernière théorie est le modèle comportemental, appelé également l'approche behavioriste, et définit "l'arriéré mental comme celui qui présente un répertoire limité du comportement." [LAMBE,78]. Et ce qui différencie les approches traditionnelles du modèle comportemental est le rôle attribué aux variables physiques, biologiques et sociales.

I.2.4. Méthodes d'intervention

Les méthodes d'intervention en déficience mentale sont, selon Sherban IONESCU, de deux types: soit médicales, soit psychologiques. Ces deux catégories sont elles-mêmes divisées en plusieurs parties. Les méthodes médicales se composent de la diétothérapie, de l'approche neurochirurgicale et de la psychopharmacologie. Les méthodes psychologiques regroupent, ce que l'auteur appelle, l'approche behaviorale, les applications de la théorie piagétienne et les psychothérapies.
Toutefois, il existe ce que Sherban IONESCU appelle les méthodes d'éducation spécialisée. Ces techniques sont l'éducation motrice, l'intervention précoce, l'éducation psychomotrice, l'éducation du langage, l'enseignement spécial, l'éducation sexuelle, l'éducation sociale, les loisirs et les activités sportives, l'insertion dans le monde du travail, la formation professionnelle, la désinstitutionnalisation et l'intégration sociale. Mais nous ne développons pas ici ces techniques car elles n'entrent pas dans le propos de ce mémoire.

* *La diétothérapie*

Sherban IONESCU [IONES,87] met en évidence la relation entre la diététique et la déficience mentale. Par exemple, un individu ayant un patrimoine génétique normal peut devenir une personne mentalement handicapée si sa diète se compose d'une certaine quantité et qualité d'aliments. Entre autres, dans ce cas, "la prescription diététique apparaît comme un acte médical justifié et nécessaire dans une approche thérapeutique globale du patient. Pourtant, à la différence d'un médicament, la diète est plus une prescription sociale qu'une prescription individuelle" [IONES,87]. En effet, les habitudes journalières du patient vont être radicalement modifiées. Or, "la nourriture est une forme symbolique de communication" [IONES,87]. Il faut dès lors s'attendre à certaines réticences de la population à l'égard de ce procédé.

* *L'approche neurochirurgicale*

Les seules déficiences mentales susceptibles d'une amélioration suffisante après le traitement chirurgical sont, selon S. IONESCU, celles développées secondairement (dont l'hydrocéphalie et les craniosténoses) et non celles existant ou constatées à la naissance. Ainsi, "l'intervention neurochirurgicale est susceptible de traiter très efficacement et très rapidement une hypertension intra-crânienne et ses redoutables conséquences" [IONES,87]. Cependant, dans certains cas, des séquelles ne peuvent être évitées.

* *La psychopharmacologie*

"L'utilisation, chez les déficients mentaux, des médicaments psychotropes -substances prescrites pour provoquer des modifications cognitives, comportementales ou émotionnelles- est devenue très répandue au cours des trente dernières années" [IONES,87]. Cette thérapie permet, entre autres, de contrôler l'hyperactivité et l'agressivité, et d'améliorer le fonctionnement cognitif. Cependant, même si ces médicaments peuvent avoir un effet

thérapeutique positif, de plus en plus de personnes hésitent à utiliser la pharmacothérapie en guise de traitement. Et ce, essentiellement parce que les différentes études réalisées se sont avérées être incorrectes sur le plan méthodologique, ou tout au moins, la valeur de leur résultat a diminué sensiblement par la présence de lacunes méthodologiques.

* *L'approche comportementale*

Afin d'évaluer la compétence sociale des personnes handicapées mentales, des échelles comportementales ou développementales sont de plus en plus utilisées. Ces échelles permettent de tester la présence de plusieurs comportements survenant durant le développement normal. "Les différences concernant les intervalles d'âge où apparaissent ces comportements et celles relatives aux aspects du développement qui y sont plus spécifiquement examinés expliquent la diversité des échelles disponibles" [IONES,87]. Les intervalles d'âge pour les échelles les plus utilisées sont de 0 à 3 ans pour l'échelle P.P.A.C., de 0 à 4 ans pour l'échelle S.E.E.D., de 0 à 6 ans pour les grilles d'évaluation comportementale de Beaudoin et Portage, et de 0 à 8 ans pour l'échelle de développement de Harvey. Elle sera choisie en fonction de l'âge chronologique de l'individu et de son niveau de développement. Par exemple, la S.E.E.D. évalue 720 comportements appartenant à 8 secteurs de développement et qui se manifestent entre 0 et 4 ans. Ces secteurs sont les domaines des motricités globale et fine, du socio-affectif, de la réception du langage, de l'adaptation et du raisonnement, de l'expression verbale, de l'alimentation, de l'habillage et de l'hygiène. A partir des résultats obtenus lors de l'évaluation du niveau de développement global, telle ou telle intervention est alors mise en oeuvre.

La programmation est, selon S.IONESCU, l'organisation d'une suite d'actions à accomplir en vue de permettre, à la personne mentalement déficiente, d'acquérir les comportements nécessaires à son adaptation. Les étapes de ce processus (qui se présente sous la forme d'une boucle) sont dans l'ordre l'évaluation, le choix des priorités éducatives, la détermination des objectifs comportementaux, la planification des apprentissages et la mesure

des apprentissages. Après cette dernière mesure, l'étape suivante est soit la première, la troisième ou la quatrième.

* *La théorie piagétienne*

Ce sujet "risque fort de soulever un tollé de la part des intervenants qui ont un préjugé contre la théorie et, aussi, de ceux qui refusent de s'intéresser aux applications pratiques de la théorie piagétienne" [IONES,87]. C'est pourquoi, notre intérêt n'étant pas d'entrer dans un débat relatif à l'application ou la non application de cette théorie, nous nous limitons à une description sommaire de la théorie de Piaget. Cet épistémologue définit une séquence de stades développementaux par lesquels tout individu doit passer. Il montre que l'ordre de succession des acquisitions est constant et que chaque palier se caractérise par une structure d'ensemble. "Il commence par un niveau de préparation et se termine par un palier d'équilibre -le niveau d'achèvement-. Les structures construites à un âge donné deviennent partie intégrante des structures de l'âge suivant" [IONES,87]. Nous en concluons donc que le développement cognitif chez l'enfant est un processus continu et graduel. Les paliers, de complexité grandissante, intègrent les précédents. Cependant, en ce qui concerne les personnes ayant une déficience mentale, l'acquisition nécessaire pour passer d'un stade à un autre est plus lente que chez une personne dite normale.

* *Les psychothérapies*

L'utilisation de psychothérapies avec les personnes handicapées mentales peut viser 8 buts: "1. vérifier, si nécessaire, le diagnostic de déficience mentale; 2. pallier l'immaturité affective; 3. réduire l'anxiété et l'agressivité; 4. améliorer la communication; 5. améliorer le comportement et les relations interpersonnelles; 6. faire accepter le déficit et les limites qui en découlent; 7. valoriser le déficient et améliorer son estime de soi; 8. entraîner le déficient à chercher adéquatemment l'aide nécessaire lorsqu'il rencontre des problèmes qu'il ne peut résoudre" [IONES,87]. Afin d'atteindre un

de ces objectifs, plusieurs méthodes et techniques sont mises à la disposition des thérapeutes. La sélection du moyen à appliquer lors de l'intervention se fait selon 3 critères: le degré d'intervention du thérapeute, le nombre de sujets impliqués dans l'intervention, et le type de communication (verbale ou non verbale) utilisé. Les différentes psychothérapies disponibles sont, entre autres, la psychanalyse, la thérapie par le jeu, le psychodrame, la musicothérapie, l'art-thérapie, les psychotérapies auprès des parents des individus mentalement déficients, la thérapie existentialiste, la technique du feed-back audio-visuel,...

Chacune de ces interventions doit faire l'objet, périodiquement, d'une évaluation afin d'établir si le traitement entrepris est efficace ou non. En réalité, cette évaluation se fait au travers d'activités d'apprentissage qui permettent de programmer les étapes de la thérapie à mener. Mais c'est fondamentalement en fonction de la problématique (du moment) du patient que doit se faire le choix de la méthode à appliquer. La thérapie alors utilisée peut être remplacée par une autre, afin de suivre l'évolution de l'individu.

I.3. CREATIVITE ET HANDICAP MENTAL

Norbert Heintz [HEINT,88] définit la créativité au niveau de l'individu mentalement déficient et de son rapport avec la société comme suit: "La créativité est une expérience vécue qui devient un signe de vie par lequel la personne handicapée mentale se reconnaît, se définit, se réalise et se valorise dans les lieux où se débattent les aptitudes et les sensibilités et où son besoin d'égalité, d'identité, de reconnaissance et de réciprocité peut s'inscrire dans le tissu social." [HEINT,88]. Toujours selon N. Heintz, la créativité peut être considérée, premièrement, comme un "autre langage" et, deuxièmement, comme "un acte social et culturel".

En premier lieu, la créativité permet de se réaliser dans un ensemble d'expériences et devient, en même temps, un autre moyen de communication où pourra surgir tout potentiel expressif qui existe à l'intérieur de la personne. Elle prolonge et complète tout ce que la personne handicapée mentale n'arrive pas à exprimer

verbalement. Elle fait apparaître d'autres "langages" procurant à la personne handicapée mentale un moyen adéquat de se libérer de tout ce qui la limite, la sépare et l'exclut du monde des personnes dites normales. La créativité crée des langages nouveaux chez les personnes mentalement handicapées: "Les langages créatifs ont, comme tout autre langage, une valeur de communication et de dialogue. (...)Pour les personnes handicapées mentales, le langage créatif n'est pas seulement l'exercice d'une quelconque technique mais surtout un moyen de communication et d'expression, un dialogue avec celui qui regarde. C'est un langage plein de messages, plein d'interactions qui dégage des émotions des angoisses, des plaisirs des contrastes." [HEINT,88]

En second lieu, la créativité assure, chez les personnes mentalement déficientes, une fonction socio-culturelle de première importance: "Etre créatif est un acte social, car la créativité est inhérente au fait de vivre." [HEINT,88] La créativité, de par sa propriété d'interactivité, est branchée sur le social. En effet, l'interactivité implique un mouvement vers l'autre, vers la communauté, vers le monde. Elle peut être considérée comme l'organe de relation organisme-environnement. Heintz considère qu'il faut chercher le sens de la créativité dans la recherche et la valorisation des attributs spécifiques à tout être humain et communs à nous tous. C'est la raison pour laquelle il la considère comme la meilleure discipline de participation pour nos "concitoyens handicapés".

La notion de créativité est, selon N. Heintz, fortement liée à celle de l'art. Nous considérons, dans notre société, que l'individu créateur doit posséder une sensibilité particulière pour exprimer ce qu'il ressent, ce qu'il souhaite représenter. En particulier, nous estimons, souvent à tort, qu'il doit être intelligent. Par là, nous éliminons (un peu trop facilement) ceux qui ne possèdent pas cette intelligence. Ainsi, les personnes mentalement déficientes ne rentrent elles pas, en général, dans nos cadres de référence et nous refusons, par conséquent, tout contenu artistique à leurs oeuvres.
Face à cette attitude trop "réductrice", il convient de présenter le problème de façon plus "ouverte". La personne ayant une déficience mentale manifeste, par toute autre forme que le langage verbal, une

intelligence différente de celle généralement admise par la société (et qui, nous l'avons vu, ne rencontre pas toujours une réelle compréhension). De plus, l'expérience montre que cette personne peut s'épanouir dans l'expression de son identité ou de sa subjectivité. C'est par l'activité créatrice qu'il lui est possible d'affirmer sa raison d'être.

Il faut savoir qu'une personne handicapée mentale peut transmettre un message et peut également en recevoir. L'expression et le vécu émotionnel ne sont pas pour autant déficitaires chez cette personne. Le problème provient du fait qu'il y a déficit, non pas au niveau du message lui-même, mais bien au niveau du **décodage** et de l'**encodage** du message.

Le domaine de la créativité ne se limite pas uniquement au monde des personnes dites normales car elle n'est liée à aucune notion de niveau de QI. Par exemple, des personnes possédant un QI très élevé peuvent faire preuve d'un manque total de créativité, alors qu'un individu considéré comme mentalement déficient peut étonner par son imagination créative. De plus, cette créativité peut se manifester sous des formes fort diverses et dans des domaines très variés. En effet, elle ne se manifeste pas que dans le domaine artistique. Ainsi, résoudre un problème rencontré dans la vie quotidienne de façon personnelle et originale révèle chez l'individu une autre forme de créativité. L. De Brabandere affirme qu'être créatif, "ce n'est pas seulement trouver des solutions originales, c'est aussi inventer de nouvelles questions." [BRABA,89].

En conclusion, il n'est pas possible de définir de façon précise et unique le concept de créativité. Celle-ci peut se manifester dans de nombreuses et diverses disciplines, chacune amenant sa propre façon de la "définir" en fonction de son expérience et de ses intérêts.
De plus, le problème de la définition de la créativité n'est pas simplifié si nous nous limitons à l'observation de la population des personnes ayant une déficience mentale. Face à une oeuvre produite par une de ces personnes, il n'y a pas toujours unanimité pour affirmer que cette oeuvre révèle, ou non, une forme de créativité.

I.4. GRAPHISME ET HANDICAP MENTAL

L. Boulangé et J.-L. Lambert, dans leur ouvrage "Les Autres" [BOLAM,81], considèrent que **"l'art brut"**, par sa spontanéité et son inventivité débranchée de l'art coutumier, est le meilleur moyen de définir les oeuvres des personnes handicapées mentales. Par "art brut", on entend "des productions de toute espèce -dessins, peintures, broderies, figures modelées ou sculptures, etc.- présentant un caractère spontané et fortement inventif, aussi peu débitrices de l'art coutumier ou des poncifs culturels, et ayant pour auteurs des personnes obscures, étrangères aux milieux artistiques professionnels" [THEVO,75]. Les oeuvres graphiques des personnes ayant une déficience mentale peuvent donc être considérées comme de "l'art brut". Cependant, cet art possède une autre orientation, chez les personnes mentalement handicapées, de celle des artistes de l'art brut en général. Il s'agit, toujours selon L. Boulangé et J.-L. Lambert, de le considérer comme un moyen d'apprentissage et d'intégration sociale pour les personnes mentalement handicapées [5].

Pour analyser le graphisme et son évolution chez les personnes ayant une déficience mentale, il serait trop simple de se baser uniquement sur l'évolution observée chez l'individu dit normal, compte tenu, évidemment, d'un important retard [6]. L'observation attentive du dessin de ces personnes procède de l'analyse simultanée de trois facteurs: la maîtrise des outils, le développement affectif et l'expérience des sujets.

* *La maîtrise des outils*

En général, les personnes mentalement déficientes, quel que soit leur niveau de développement intellectuel, présentent d'importants retards dans l'acquisition des capacités motrices et sensorielles. "Or c'est précisément sur l'organisation sensorielle et motrice que se basent les productions graphiques. (...)Incontestablement,

---

[5] cfr. infra

[6] cfr. les 4 niveaux de l'évolution graphique chez l'enfant, dégagés par Osterrieth [OSTER,76].

l'expression graphique a un rôle à jouer dans les techniques d'apprentissage du contrôle sensoriel et moteur.» [BOLAM,81].

* *Le développement affectif*

Chez l'enfant normal, le dessin porte la marque de la vie émotionnelle, il reflète une vue d'ensemble de la personnalité. En ce qui concerne les enfants mentalement handicapés, on ne dispose que de peu d'informations à propos des caractéristiques de leur développement affectif. Nous remarquons toutefois que les limitations imposées par le handicap de ces enfants entraînent vraisemblablement une structuration différente de l'affectivité. Il faut donc s'interroger sur le statut du développement affectif du sujet handicapé mental avant d'attacher une valeur symbolique à une de ses productions.

* *L'expérience du sujet.*

Le milieu scolaire spécialisé joue un rôle important dans l'acquisition de l'expérience des enfants handicapés. L'étude d'un dessin doit tenir compte de l'histoire du sujet, des conditions environnementales dans lesquelles il a évolué et les formes d'apprentissage qu'il a expérimentées.

Outre les valeurs d'apprentissage et d'intégration sociale que l'art brut, selon L. Boulangé et J.-L. Lambert, peut revendiquer, les productions artistisques chez les personnes handicapées mentales ont aussi une valeur interprétative.

* *Valeur d'apprentissage des productions artistiques*

Les domaines suivants sont susceptibles d'être touchés par les diverses formes d'expression artistique: la motricité générale, le développement sensoriel, la motricité fine, la coordination visuo-

motrice, l'équilibre, le langage réceptif, le langage expressif, la communication non-verbale, la socialisation,... De plus, selon L. Boulangé et J.-L. Lambert, le développement des activités artistiques chez les sujets handicapés requiert la construction d'un milieu adapté à leurs besoins.

* *Valeur d'intégration des productions artistiques*

Les personnes mentalement handicapées ne disposent pas de nombreux moyens pour affirmer leur existence. "Leurs oeuvres, éléments du musée universel de l'art brut, représentent un point d'attache avec le monde qui les entoure." [BOLAM,81]. "La normalisation des conditions dans lesquelles se place l'expression artistique des handicapés mentaux est un facteur indéniable, permettant de rompre les barrières entre les handicapés mentaux et les "normaux"" [BOLAM,81]. L'oeuvre d'art de la personne mentalement déficiente est ainsi une "ouverture relationnelle" sur le monde et elle facilite la socialisation de l'artiste. "Elle est la condition même de l'enrichissement humain et culturel de l'individu." [KAPTA,78].

* *Valeur interprétative des productions artistiques*

Il peut être intéressant d'analyser les productions graphiques des personnes handicapées mentales si l'on sait qu'elles représentent l'expression de la personnalité individuelle et de sa problématique. Cependant, l'accès à la symbolisation chez ces personnes est fort difficile, voire impossible, compte tenu de la mauvaise structuration de leurs outils d'expression symbolique. "Il s'avère que lorsqu'on connaît bien le dessinateur beaucoups d'éléments de ses productions prennent un caractère très significatif au point de vue psychologique, mais qu'il est infiniment plus hasardeux de partir du dessin pour connaître l'individu." [OSTER,76].

Hassan Kaptan [KAPTA,78] dégage 3 grands stades caractéristiques de développement du processus créatif de la personne mentalement handicapée. Ce découpage est bien entendu artificiel et les lignes de démarcation entre ces stades ne se remarquent pas comme telles.

Les différents stades sont les suivants:

- Le premier est une période de tâtonnements, pendant laquelle l'individu se cherche et se perd sans cesse.
- Le second un stade "d'interrogation" qui commence avec la fouille de ce que le premier stade a pu apporter comme éléments développables.
- Le troisième est une étape de "réalisation" qui est caractérisé, chez l'individu, par un grand intérêt pour l'aspect purement technique d'une réalisation picturale.

Pour Anne Vigliaron [VIGLI,83], l'activité graphique est un apprentissage fortement lié à l'évolution de la motricité chez l'enfant[7]. "J'ai découvert que le griboulli apparaît avec l'apprentissage de la marche et avec le sens de l'équilibre. C'est l'expression biopsychique propre à chaque individu. La précision du geste est liée à la possibilité pour les segments du membre qui l'exécute de trouver un appui suffisamment ferme dans le reste du corps.(...) Tout ceci explique la liaison intime entre le corps et l'acte graphique" [VIGLI,83]. Pour cet enfant, ce n'est pas tellement le "produit" final qui compte. "Il ne s'attache pas spontanément à son oeuvre. (...) Il concentre toutes ses énergies sur le geste du moment, seul compte le plaisir du geste, le trait actif qui se développe et vit de sa propre vie." [VIGLI,83] A cela s'ajoutent le plaisir de "l'inscription", la satisfaction de laisser une trace, de "maculer une surface".

En conclusion, la production artistique implique la mise en oeuvre de processus psychologiques, tant au niveau cognitif qu'affectif. Quant à l'oeuvre produite, elle est le moteur d'un système de communications et de relations. Les techniques utilisées ainsi que leur maîtrise par celui qui réalise l'oeuvre, influencent la

---

[7] mentalement handicapé ou non.

spécificité et la qualité d'une production artistique. Ce sont là les trois dimensions (individuelle, relationnelle, technique) que M. Mercier et O. Witdouck [MERWI,91] dégagent de l'activité création artistique; ces trois dimensions étant "toujours traversées par les déterminants culturels d'une part, et la recherche de plaisir d'autre part" [MERWI,91].

L'artiste mentalement handicapé apportera sa spécificité dans la production artistique; cette spécificité étant liée au déficit cognitif qu'il doit assumer et qui modifie le fonctionnement du système d'interactions. Cette carence ne l'empêche toutefois pas de "faire oeuvre d'art" et, par son travail artistique, de révéler "les déterminants profonds des productions artistiques chez tout homme" [MERWI,91].

I.5. INFORMATIQUE, HANDICAP MENTAL ET CREATIVITE

Dans le point abordé précédemment, nous avons décrit la démarche créative, plus particulièrement dans le domaine du graphisme, chez la personne ayant un handicap mental. Nous décrivons dans ce point l'impact que peut avoir l'outil informatique sur le système de créativité des personnes mentalement handicapées. Cependant, il est d'abord intéressant de se pencher sur les caractéristiques qui peuvent rendre cet outil attrayant auprès de ces personnes et sur ce qui le rend intéressant dans l'activité de création artistique, plus précisément dans le graphisme.

Trois caractéristiques paraissent être pertinentes: premièrement, l'ordinateur peut accroître les performances; deuxièmement, l'ordinateur peut fournir un moyen d'accès pour les personnes handicapées qui ne peuvent pas utiliser les matériaux traditionnels; troisièmement, l'ordinateur offre l'opportunité à l'expérimentation et la croissance intellectuelle qui peut mener directement ou indirectement vers d'autres activités.

* *Accroître les performances*

La créativité est étouffée lorsque l'artiste ne peut effectuer un effet désiré. Ainsi, l'utilisation libre de couleurs et de mouvements dans le dessin ne nécessite pas nécessairement l'utilisation d'un ordinateur, mais l'expérience des rythmes générés par une répétition soigneuse de "motifs" particuliers peut être difficile à appréhender pour un artiste non averti travaillant à main libre. De plus, le fait d'avoir un ordinateur parmi les moyens d'expression disponibles permet à l'artiste d'avoir à sa disposition un nouveau moyen de communication. La vitesse contribue aussi à améliorer les performances de l'utilisateur. Sans devoir attendre que leurs capacités motrices se développent, les plus jeunes utilisateurs peuvent appliquer leurs capacités cognitives et tester leurs idées. Ainsi, l'ordinateur permet le développement très rapide de leurs capacités graphiques.

* *Un moyen d'accès pour les personnes handicapées*

Cette caractéristique découle en partie de la précédente. Grâce à l'ordinateur, certains utilisateurs incapables d'écrire la moindre lettre peuvent réaliser des oeuvres artistiques relativement sophistiquées et esthétiques, concevoir des appareils mécaniques, dessiner des cartes, concevoir la chorégraphie d'une danse ou même composer une symphonie. Il faut cependant que ces utilisateurs parviennent à maîtriser "l'outil informatique" mis à leur disposition et disposent des capacités requises pour la création d'oeuvres originales.

Pour l'utilisateur dont le problème majeur est la distraction, cet outil peut offrir un autre moyen d'accès subtils. Cet utilisateur se retrouve souvent privé de certaines informations lorsque des évènements extérieurs et étrangers interfèrent avec son attention. Ici, la capacité de l'ordinateur à accroître les performances peut servir à

mieux focaliser l'attention. En accélérant l'interaction, il laisse moins de place pour la distraction.

* *Une opportunité pour l'expérimentation*

L'art n'est généralement pas considéré comme étant une activité de "recherche", mais il possède la vertu d'être sans limite et est, pour de nombreux pratiquants, fort attractif. Pour le véritable artiste, la composition peut être guidée par d'authentiques règles et contraintes -parfois auto-imposées-. Mais les débutants ont tant de difficultés à manipuler l'outil que l'expérimentation dépasse de loin leurs possibilités. La capacité qu'a l'ordinateur à réduire le temps et l'habileté technique nécessaire à l'exécution d'une composition graphique rendent possible à un débutant d'effectuer de véritables expériences dans ces domaines attractifs, en imposant des règles et des contraintes, et en utilisant les résultats pour guider de nouvelles explorations.

M. Mercier et O. Witdouck [MERWI,91] essayent de définir l'informatique comme un **outil culturel** et tentent de mieux comprendre la personne handicapée face aux nouvelles technologies. Ils considèrent qu'en révolutionnant les moyens techniques de mémorisation, de traitement et de communication des informations, l'ordinateur est susceptible de modifier profondément les interactions de la personne handicapée créatrice avec elle-même, de son oeuvre et son environnement. "L'ordinateur et l'informatique présentent des caractéristiques techniques qui déplacent les limites dans les contraintes de productions." [MERWI,91].
Pour ce faire, ils analysent les caractéristiques essentielles de l'ordinateur et du traitement de l'information par l'ordinateur dans le domaine de la production artistique. Pour eux, l'ordinateur:

- permet d'accélérer la production d'oeuvres en effectuant les tâches répétitives et dévoreuses de temps comme, par exemple, le remplissage automatique d'une zone en une couleur donnée.
- peut jouer un rôle actif dans la production artistique en suscitant la créativité. Ainsi, l'utilisation de formes géométriques prédéfinies peut faire apparaître de nouvelles idées pour l'élaboration de la production artistique.

- peut conserver et dupliquer aisément de l'information. Il devient donc possible "d'agir" sur une copie d'une oeuvre déjà réalisée sans pour autant provoquer d'irrémédiables dégâts à l'oeuvre originale.

En ce qui concerne la création graphique au moyen de l'informatique, M. Mercier et O. Witdouck avancent que l'ordinateur offre des possibilités graphiques, en permettant, entre autres, de créer aisément des dessins de haute précision. Mais, "même si certaines productions n'ont que peu de valeur artistique, elles procurent à la personne mentalement handicapée le plaisir de réaliser un dessin conforme à ses attentes." [MERWI,91].
Ils dégagent ainsi deux nouveautés par rapport aux techniques anciennes:
* L'échec n'étant plus pénalisé, l'auteur peut tester d'autres possibilités et ainsi prendre confiance en lui.
* L'ordinateur peut devenir un outil pédagogique pour une initiation esthétique.
Mais, c'est seulement après un passage d'apprentissage de la technique informatique que l'ordinateur pourra devenir le lieu de créations artistiques plutôt que de réalisations picturales.

En conclusion, l'ordinateur est un outil intéressant pour une personne créatrice. "Il peut offrir le confort que la créativité revendique" [BRABA,89]. Il peut être un outil pour susciter des idées nouvelles, mais il n'est pas un "remède" contre le manque de créativité dont pourraient faire preuve certains de ses utilisateurs. "Cette machine est certe considérée depuis toujours comme le degré zéro de la créativité (...) Il est peut-être paradoxal de constater que l'incapacité de créer d'un ordinateur en fait un outil tout indiqué pour la créativité humaine. Mais il reste bien sûr limité par son absence de force motrice et il s'apparente à un bras de levier». [BRABA,89]. L'ordinateur, en tant que bras de levier, n'a donc un effet multiplicateur des forces de la créativité que si il est manipulé par une personne capable de s'en servir et disposant des forces créatrices requises. Ainsi, seules les personnes ayant un degré de créativité suffisamment élevé et possédant une bonne maîtrise de l'ordinateur verront dans celui-ci un outil idéal pour exprimer leur

créativité propre. M. Mercier et O. Witdouck concluent que "l'ordinateur comme outil de créativité n'est qu'une technique supplémentaire mise au service de la personne mentalement handicapée, mais nous devons rester bien conscient qu'elle ne pourra s'avérer bénéfique que pour une certaine part de cette population." [MERWI,91]

I.6. LA BANDE DESSINEE

La bande dessinée est née, presque clandestinement, en Suisse, vers 1827. Depuis le début des années 70 de ce siècle, le public a pu constater un considérable développement de la narration figurative. Elle n'a cessé de s'affirmer comme un art graphique et est devenue, à ce titre, le **"Neuvième Art".** Elle s'est développée à un point tel qu'elle est, aujourd'hui, un moyen d'expression et de communication très important, sinon des plus importants. Le "dernier changement fondamental, celui du statut de la bande dessinée, est qu'elle est désormais reconnue comme un des plus importants moyens d'expression du siècle, à tel point qu'elle est utilisée par tout le monde: philosophes, enseignants, pédagogues, politiciens, éditeurs,.." [UNIVE,90]. Elle est devenue un des moyens de communication et d'expression les plus utilisés, et permet à l'artiste d'exprimer sa créativité.

I.6.1. Définition, caractéristiques et fonctionnement

"La bande dessinée est une forme de récit fondé, comme dans un film, sur une harmonie de l'image et du son. Ce récit est fait au moyen d'images dessinées (à la différence du roman-photo), fixes (à la différence du dessin animé), à l'intérieur desquelles figurent les sons: bruits, commentaires, dialogues; ces derniers s'inscrivent en général dans une réserve blanche aux contours irréguliers, dénommée, en français, ballon, bulle ou phylactère" [UNIVE,90]. Le texte des bandes dessinées peut être dans l'image, hors de l'image ou même sur l'image.

Tout aussi importantes sont la notion de continuité (ou de séquence), les dimensions des cadres contenant les images et, facultativement, du son (ou plutôt, devrions-nous dire, la représentation graphique ou/et littéraire du son), les couleurs utilisées, la représentation de la bulle et de son contenu, et les idéogrammes.

C'est le développement de toutes ces caractéristiques qui a permis à la bande dessinée d'obtenir ses lettres de noblesse et de devenir un moyen d'expression très important appelé le "Neuvième Art".

* *La notion de continuité*

La plupart des bandes dessinées se présentent sous forme d'images qui se succèdent sur le plan horizontal et de haut en bas. Mais ceci n'empêche nullement que des cadres soient disposés, par exemple, en diagonale.

* *Les cadres*

La dimension et le cadrage des cadres donnent une certaine signification aux bandes dessinées. Comme cadrage, il existe, entre autres, les plans d'ensemble, général et, le gros plan. Chacun de ces plans possédant, de par sa nature même, une charge affective ou significative.

* *La couleur*

Les couleurs de l'image sont choisies harmonieusement, avec beaucoup de soin par l'auteur selon le style de sa bande dessinée (humoristique, dramatique, historique, fantastique,...): c'est le rôle esthétique et (surtout) fonctionnel de la couleur. Mais elle peut également avoir un rôle psychologique. Comme, par exemple, le fait

de sélectionner certaines couleurs en fonction du lecteur qui, à la vue de l'image, se projette en l'adoptant ou en la rejetant.

* *La bulle et son contenu*

La bulle est, en elle-même, l'outil qui a permis et permet de restituer l'essentiel du code linguistique que ce soient les paroles, les dialogues, les monologues, les onomatopées, les pensées, les bruits,... De par sa représentation, elle permet de faire passer le ton du dialogue. Ainsi, une bulle ornée de stalactites révèle la froideur du ton, un ballon tracé en dents de scie indique que la voix est retransmise (téléphone,radio...), une bulle provenant de la tête traduit une pensée.
Quant à son contenu, il permet de restituer l'intensité sonore du ton du dialogue ou du monologue, les bégaiements, la mélodie, les soupirs, les bruits,... Ainsi, de grosses lettres expriment le hurlement. Les bruits, eux, donnent lieu à un réalisme phonétique. Par exemple, une automobile émettra un "vroââr" ou un "rac pout pout" selon la marque.
Quant à leur localisation, les ballons sont placés sur l'image (ou dans le cadre) de telle manière qu'ils ordonnancent la réplique des acteurs des bandes dessinées, acteurs qui "utilisent" le style direct[8] pour communiquer.
Mais si le texte a un rôle de communication dans le dessin, il est également "graphique": il fait partie de l'esthétique du dessin (de son graphisme), ne serait-ce que par la taille des caractères.

* *Les idéogrammes*

Un idéogramme est un signe représentatif d'une idée; c'est un moyen de communication "économique". Il contribue à renforcer le sentiment suggéré. Il existe comme idéogrammes, entre autres, les étoiles, les éclairs, les ampoules électriques, les nuages noirs,... qui, de plus, peuvent être combinés ensemble.

---

8 Les acteurs parlent en leur nom propre.

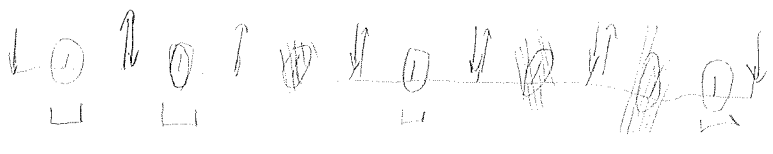
Cependant, en plus d'une utilisation adéquate de ces caractéristiques[9] et du fait que la Bande Dessinée est avant tout un "regroupement" d'images, elle met en évidence deux relations importantes quant à sa compréhension générale. Celles-ci sont la relation entre les différentes images et celle existant entre une image et son texte.
Toutefois, la bande dessinée est, en premier lieu, un agencement d'images réalisées par l'auteur et par laquelle il exprime sa créativité. Chaque image est dessinée selon l'information et l'esthétique que l'auteur veut faire passer à travers le graphisme (de la B.D.).

En ce qui concerne la relation inter-images, Fresnault et Deruelle nous disent que "les images isolées signifient dénotativement peu sans leurs tenants et leurs aboutissants. C'est le sens général né d'une succession qui détermine à chaque instant la valeur d'une image" [FRESN,72].
Quant à la relation entre l'image et le texte, ces auteurs affirment que lire des bandes dessinées c'est lire doublement. Le lecteur décode une image et (normalement, si il existe) un texte. L'appréciation globale de ces deux derniers éléments permettant d'appréhender le contenu du cadre. Et plus la complémentarité entre les éléments graphique et narratif est grande, plus les bandes dessinées sont réussies. Mais pour comprendre cette relation, il ne suffit pas de regarder l'image et de le lire le texte, il faut aussi un acte d'abstraction, une démarche intellectuelle pour les mettre en relation.

I.6.2. Importance de la BD pour les personnes handicapées mentales

La création de bandes dessinées peut devenir un excellent moyen d'expression pour les personnes mentalement déficientes. Elle peut apporter à cette population de nombreux avantages mais elle

9 présentées ci-avant.



requiert, par la même occasion, certaines capacités dont ne dispose pas l'entièreté de cette population.

I.6.2.1. Apports de la création de bandes dessinées pour les personnes mentalement déficientes

Le principal avantage de la création de bandes dessinées par des personnes mentalement déficientes concerne la communication. La phase même d'élaboration de la bande dessinée est, selon L. Boulangé[10], un moyen de communication idéal mis à la disposition des personnes ayant une déficience mentale. En effet, la réalisation d'une page de bandes dessinées peut se faire par l'intermédiaire d'un jeu de communication entre la ou les personnes mentalement handicapées et l'animateur (qui pourrait se charger d'introduire dans le dessin le texte "dicté" par la personne mentalement déficiente), ainsi qu'entre les personnes mentalement déficientes elles-mêmes. C'est véritablement "leur histoire" que ces personnes racontent de façon structurée dans la page de bandes dessinées.
De plus, la page de bandes dessinées rend plus aisée la transmission du "message" que ne le permet une suite de dessins réalisés sur des pages distinctes. En effet, une fois la page imprimée, l'histoire qui y est racontée peut être visualisée d'un seul coup d'oeil.

Un autre avantage de la création de bandes dessinées concerne la focalisation de l'attention du créateur sur son oeuvre. La réalisation d'une page de bandes dessinées se différencie de celle d'un simple dessin par le fait qu'une plus grande persévérence sera demandée à ces personnes pour structurer leur message sur une page de BD plutôt que sur un seul et unique dessin. Il leur faudra, si nécessaire, retravailler la planche de BD, à plusieurs reprises, afin de la terminer. La bande dessinée force ainsi la canalisation de l'attention de la personne mentalement handicapée sur un seul sujet (souvent choisi par elle), et cela pendant une durée plus longue que celle nécessitée pour la réalisation d'un simple dessin.

---

10 animateur au centre CREAHM de Liège

La réalisation d'une bande dessinée peut posséder un aspect très motivant et valorisant aux yeux de l'entourage du créateur. Cette réalisation peut, en effet, faire partie d'un projet visant à rassembler différentes pages de BD en un bouquin lequel pourrait être dupliqué et distribué.

La création de pages de bandes dessinées apporte un nouveau souffle de créativité. Une page de bandes dessinées peut être vue comme un rassemblement sur une seule planche de différents éléments (dans ce cas-ci, des dessins) qui pris séparément n'auraient pas de réelle valeur créative. La page de bandes dessinées donne ainsi un sens à des dessins disparates en les réunissant en fonction d'un thème choisi.

I.6.2.2. Capacités requises pour la création de bandes dessinées

Bien entendu, l'activité concernant la création de bandes dessinées ne s'adresse pas à toute la population des personnes ayant un handicap mental. Il leur est non seulement demandé de faire preuve de créativité, mais aussi de posséder des capacités graphiques leur permettant de réaliser une oeuvre graphique, en plus d'avoir un grand besoin de communiquer et de s'exprimer. La notion de séquencement d'évènements dans le temps doit également être bien comprise pour pouvoir réaliser un scénario qui servira de base à la création de bandes dessinées.

Il serait intéressant de voir une page de bandes dessinées réalisée par un groupe où chacun apporterait sa spécificité. Ainsi, celui ou celle dont les capacités graphiques sont les plus élaborées pourrait s'occuper de la réalisation graphique, tandis qu'un(e) autre s'occuperait de raconter le sujet.

I.6.3. L'apport de l'ordinateur

Pour certaines personnes ayant un handicap mental, l'utilisation de l'ordinateur peut être en elle-même une activité très motivante. Un logiciel spécialisé dans la conception de bandes dessinées devrait permettre de constituer une page de bandes dessinées via un périphérique d'ordinateur adapté aux capacités de l'utilisateur et d'imprimer cette page sur une feuille de papier [11].

L'ordinateur, de par ses capacités de conserver des informations sous divers aspects, est un excellent "réservoir" pour ces éléments disparates que sont les dessins réalisés par les personnes mentalement handicapées ou d'autres images provenant du "monde" qui les entoure (par exemple, des photos digitalisées). Il pourrait être un outil permettant de choisir quelques dessins dans cet ensemble et de réaliser un "tout cohérent".

---

11 Cette page pourrait être retravaillée, selon L. Boulangé, à la main, par les personnes mentalement handicapées.

I.7. CONCLUSIONS

Ce chapitre permet de se rendre compte qu'il est relativement difficile de définir, de façon précise et unanime, les concepts de handicap mental et de créativité tant ceux-ci sont difficiles à appréhender et incluent bon nombre de facettes diverses. C'est ainsi que nous ne disposons pas de modèle théorique des utilisateurs potentiels du logiciel de création de bandes dessinées. Nous avons fait apparaître dans ce chapitre certaines des caractéristiques que l'utilisateur mentalement déficient doit posséder pour pouvoir s'adonner à l'activité de création de bandes dessinées en plus de celles exigées par la manipulation de l'ordinateur. Ainsi, l'élaboration d'un logiciel permettant à des personnes mentalement déficientes d'exprimer leur créativité par le biais de la création de bandes dessinées ne s'est pas faite sur des bases théoriques bien définies. C'est la raison pour laquelle elle ne fut pas très aisée et a nécessité une longue période de "tâtonnement".
Finalement, ce chapitre montre qu'il est permi de croire que la micro-informatique peut trouver sa place dans le domaine précis de la créativité des personnes mentalement déficientes et offrir un nouvel "outil" pour tenter de mieux les comprendre.

Chapitre II

Description du projet "Logiciel BD"

II.1. INTRODUCTION

Le développement d'un logiciel nécessitant une analyse préalable des désirs des demandeurs, nous présentons dans ce chapitre le début de l'élaboration du cahier de charge du projet "Logiciel BD". Nous y décrivons un énoncé, de sa définition initiale à sa version définitive, qui a pour objectif de déterminer de la façon la plus précise possible ce qu'il s'agit de réaliser. Nous présentons, dans un premier temps, l'énoncé très sommaire de départ qui a servi de base à l'élaboration du projet. Viennent ensuite les différents choix effectués et les ajouts réalisés à l'énoncé de départ durant son élaboration pour arriver à la version finale telle qu'elle a été définie à la suite de différents contacts avec les acteurs[1] concernés ainsi que des observations réalisées "sur le terrain". Nous terminons par une description des différents composants du logiciel. Le cahier de charge sera complété, au chapitre IV, par une analyse de la structuration des

[1] psychologues, animateurs d'ateliers pour personnes mentalement déficientes et les personnes handicapées mentales elles-même.

informations et des fonctionnalités du logiciel ainsi, qu'au chapitre VII, par une analyse de son interface avec l'utilisateur.

II.2. ENONCE DE DEPART ET GENESE DU PROJET

Au départ, les spécifications du projet de réalisation d'un logiciel permettant à des personnes mentalement déficientes de créer des pages de bandes dessinées étaient assez réduites. Il a donc fallu déterminer les spécifications en respectant les objectifs que poursuit le projet et qui ont été soulevés au chapitre I.

L'énoncé de départ était formulé comme suit: "Le logiciel à élaborer, doit permettre à des personnes souffrant d'un handicap mental de créer des bandes dessinées. Les images servant à élaborer les différents dessins contenus dans ces bandes dessinées peuvent provenir de sources distinctes (caméra vidéo, scanner d'images, logiciels de création graphique, ...) pour être retravaillées par la suite. Ces personnes doivent aussi avoir la possibilité d'imprimer les pages de bandes dessinées qu'elles ont réalisées".

C'est sur cet énoncé de base que sont venu s'ajouter, par la suite, des spécifications plus précises. Celles-ci ont été précisées à la suite d'entrevues avec des animateurs et psychologues travaillant dans le domaine du graphisme avec des personnes ayant une déficience mentale ainsi qu'avec des animateurs utilisant l'ordinateur lors de ces scéances.

En plus des observations que nous avons pu faire lors des scéances sur ordinateur, nous avons pu expérimenter l'utilisation d'un logiciel de création de bandes dessinées avec des personnes handicapées mentales dans des ateliers créatifs. Le logiciel en question a été conçu pour l'Amiga 500 et 2000 et peut se trouver dans le commerce sous le nom de "ComicSetter". Bien que n'étant pas destiné à des personnes mentalement déficientes, il nous a apporté différentes idées que nous avons tenté d'intégrer à l'énoncé en collaboration avec les animateurs des ateliers.

Suite à l'utilisation de "ComicSetter" avec des personnes mentalement déficientes nous avons été convaincu de la nécessité de créer un nouveau logiciel qui soit plus adapté à leurs besoins. En effet, ce logiciel faisait preuve de nombreuses carences en ce qui concerne

l'interface avec l'utilisateur et nous avons alors estimé qu'il était peu adapté à la population des personnes mentalement déficientes.

Pour pouvoir faire évoluer le projet d'élaboration d'un nouveau logiciel de création de bandes dessinées, il nous a fallu, dans un tout premier temps, nous intéresser à la population cible afin d'en connaître les caractéristiques et les capacités intellectuelles[2]. Ensuite il a fallu déterminer quelles seraient les possibilités que devrait ou ne devrait pas offrir le nouveau logiciel de création de bandes dessinées en tenant compte, bien entendu, des caractéristiques de cette population.

Ces sujets de réflexion ont guidé nos axes de recherches, d'observations et de discussions. C'est à la suite d'un stage en Angleterre où nous avons rencontré des personnes concernées par l'élaboration de logiciels destinés à l'éducation que nous avons mis au point notre énoncé final. Celui-ci est décrit au point suivant.

II.3. ELABORATION DE L'ENONCE FINAL DU PROJET

Nous présentons ci-après les éléments qui ont été ajoutés à l'énoncé de départ pour consitituer, avec celui-ci, l'énoncé final du logiciel. Chacun de ces éléments a été défini suite aux observations réalisées dans des ateliers pour personnes mentalement déficientes et de discussions avec des psychologues et des animateurs d'ateliers créatifs. C'est ainsi que chaque point de l'énoncé peut trouver une justification fondée sur l'expérience ou basée sur des recommandations de personnes compétentes que nous avons rencontrées.

Enoncé final

L'utilisateur mentalement déficient, vu son handicap, ne peut manipuler seul le logiciel. Il doit être aidé tout au long de la manipulation par une autre personne maîtrisant le logiciel, en l'occurence l'animateur de l'atelier créatif. Le rôle de ce dernier ne se limite pas uniquement à superviser et à aider l'utilisateur mentalement déficient en corrigeant ses

---

[2] que nous avons développé au chapitre I.

actions, son rôle est aussi de "préparer" le logiciel avant toute utilisation par une personne mentalement handicapée afin d'adapter le mieux possible "l'outil" aux besoins de chaque utilisateur.

Afin de ne pas trop perturber les utilisateurs, le logiciel doit respecter une certaine simplicité du point de vue de son utilisation. Pour ce faire, un soin tout particulier doit être apporté à l'interface homme-machine[3]. C'est la raison pour laquelle nous avons opté pour l'emploi d'une souris à un seul bouton (ou bien, s'il existe plusieurs boutons sur la souris, de donner la même signification à tous les boutons). Cette décision a été prise suite aux expériences effectuées avec le logiciel "ComicSetter", lequel perturbait l'utilisateur par l'assignation de fonctions diverses à chacun des boutons de la souris de l'Amiga. Quant au clavier, il n'est disponible à l'utilisateur mentalement déficient que lorsque le l'opération exécutée nécessite l'introduction de texte. Il pourrait cependant être utilisé par l'animateur pour "court-circuiter" certaines actions réalisées au moyen de la souris, ou pour accéder à certaines fonctions "privilégiées"[4].

La création des dessins d'une page de bandes dessinées nécessite l'utilisation d'images qui sont mises à la disposition de l'utilisateur. Pour rappel, celles-ci peuvent provenir de différentes sources (images digitalisées[5], images provenant de logiciels de création graphique,...). C'est la raison pour laquelle nous avons estimé qu'il était nécessaire de rassembler ces images dans un seul et même endroit accessible par le logiciel de création de BD. Nous appelons cet endroit, **BANQUE D'IMAGES (B.I.)**. Cette banque d'images permet un accès aisé aux images qu'elle contient. Pour ce faire, elle sera organisée en sous-groupes d'images représentant chacun un thème bien défini[6]. Nous appellerons **DOMAINES** les thèmes ainsi définis. Il faudra ainsi connaître le domaine auquel apartient l'image avant de pouvoir accéder à celle-ci. La B.I. contiendra initialement des images concernant divers thèmes et qui pourront avoir été réalisées par des graphistes. Celles-ci servent de base à la réalisation des premières pages de bandes dessinées et faciliteront le travail de l'utilisateur lors des toutes premières utilisations du logiciel. La B.I. pourra, par la suite, être enrichie d'images provenant des différentes

---

3 cfr chapitre VII.

4 telles que les accès au contenu des mémoires secondaires.

5 provenant de caméras vidéos ou de scanners.

6 par exemple, les personnages, les décors, les monstres, ...

sources citées plus haut et qui auront été créées par les utilisateurs eux-mêmes.

L'ajout d'images provenant des différentes sources peut provoquer, à la longue, un accroissement considérable de la taille de la B.I. et peut, par conséquent, compliquer la tâche de sélection d'images pour un utilisateur mentalement handicapé. Nous avons ainsi décidé de créer deux banques d'images distinctes: une **PRINCIPALE** et une **SECONDAIRE.** Le rôle de la première est de centraliser toutes les images disponibles pour tous les utilisateurs. Quant à la B.I. secondaire, elle est principalement réservée à un seul utilisateur. Il n'y a donc qu'une B.I. secondaire par utilisateur. Ceci a pour but de réduire considérablement la quantité d'images directement accessibles à l'utilisateur lors de la création des dessins d'une page de bande dessinée. Ces B.I. secondaires seront garnies par l'animateur au moyen des images provenant de la B.I. principale.

La première étape lors de la création d'une page de bandes dessinées est de déterminer la **MISE EN PAGE.** La mise en page définit la disposition des cases vides sur une page blanche de dimensions données. Elle définit aussi les dimensions de chaque **CASE** rectangulaire. Le logiciel offre à l'utilisateur la possibilité de créer lui-même la mise en page qu'il désire. Cependant, afin de permettre à la majeure partie des utilisateurs handicapés mentaux de réaliser les dessins de la page dès les premières manipulations, une série de mises en pages prédéfinies seront proposées. Celles-ci peuvent être utilisées directement sans devoir recourir à une phase de création qui pourrait s'avérer, dans un premier temps, trop complexe pour une partie de la population concernée.

Après la création ou la sélection d'une mise en page, l'utilisateur peut accéder à toutes les cases de celle-ci. Il ne peut cependant travailler que dans une seule case à la fois que nous nommons **CASE DE TRAVAIL.** Nous avons choisi cette option à la suite d'expériences vécues lors des tests d'utilisation du logiciel "ComicSetter" avec des personnes handicapées mentales. Ce logiciel affichait, en effet, plusieurs cases de la page en même temps à l'écran. Certaines des cases affichées pouvaient être en partie cachées étant données les limitations de la zone d'affichage de l'écran. Face à cette situation, les utilisateurs ont eu des difficultés à retrouver la case dans laquelle ils étaient occupés à travailler. De plus, les cases partiellement affichées leur donnaient l'impression que leurs dessins

étaient en partie détruits ou avaient été coupés. Ce sont les raisons pour lesquelles nous avons décidé que le nouveau logiciel n'afficherait qu'une seule case entière à la fois.

Au début de la création d'une page de bandes dessinées, notre logiciel de création de bandes dessinées propose automatiquement la première case de la page. Par la suite, la sélection d'une nouvelle case de travail se fait à l'aide de la souris en "cliquant" sur des flèches de défilement ou en "cliquant" directement dans la case désirée de la page affichée en réduction à l'écran.

La réduction d'une des cases précédentes existantes est affichée en permanence à l'écran. Cette case est appelée **CASE MEMO.** Le rôle de la case-mémo est de permettre à l'utilisateur de visualiser le contenu des cases précédentes, donnant ainsi la possibilité à l'utilisateur de se remémorer plus facilement l'histoire en cours. Cette case permet de combler en partie les déficiences du processus mémoire des personnes mentalement déficientes.

Lors de la création d'une page de bandes dessinées, le travail de l'utilisateur consiste à remplir chaque case de la page avec des images provenant de la B.I. Secondaire après les avoir modifiées au moyen des **OUTILS GRAPHIQUES** offerts par le logiciel. L'ensemble de ces outils constitue un minimum nécessaire à la réalisation de retouches sur les images. En outre, il est possible pour l'utilisateur de dessiner directement dans le contenu des cases.

Des **PHILACTERES** (ou bulles de dialogue) ainsi que des **COMMENTAIRES** peuvent être introduits dans chaque case. Ces bulles et ces commentaires contiennent le texte qui a été introduit au clavier par l'utilisateur accompagné de l'animateur.

La page de bande dessinée en cours de réalisation ou de modification doit pouvoir être entièrement affichée à l'écran ou imprimée et ce, à la demande de l'utilisateur. De même, le **DESSIN** réalisé dans une case de la page de bandes dessinées doit aussi pouvoir être imprimé.

Etant donnés les problèmes qu'ont les personnes mentalement handicapées à concevoir graphiquement une histoire sur plus d'une page, une scéance de création de bandes dessinées se limitera à la création d'une

seule page à la fois, celle-ci ne correspondant qu'à une seule histoire racontée.

L'utilisation de la couleur pour la réalisation d'une page de bandes dessinées est souhaitée mais pas nécessaire. La préférence est donnée au noir et blanc[7] plutôt qu'à un ensemble trop réduit de couleurs dû aux limitations techniques de l'ordinateur utilisé.

II.4. PRESENTATION DES COMPOSANTS DE "LOGICIEL BD"

Nous présentons ici une description de "Logiciel BD" bâtie à partir de l'énoncé présenté dans le point précédent. Cette description correspond à une solution parmi d'autres. La solution choisie répond aux différentes exigences que nous avons précédemment énoncées en ce qui concerne la population cible mais aussi en fonction des limites techniques des ordinateurs personnels que l'on trouve actuellement sur le marché à des prix "raisonnables" pour des institutions.

Le graphique suivant présente les différents éléments qui composent "Logiciel BD" et montre la manière selon laquelle les différents composants du logiciel interagissent. Les flèches indiquent le sens dans lequel se font les échanges d'informations.

[7] en fait, des nuances de gris.

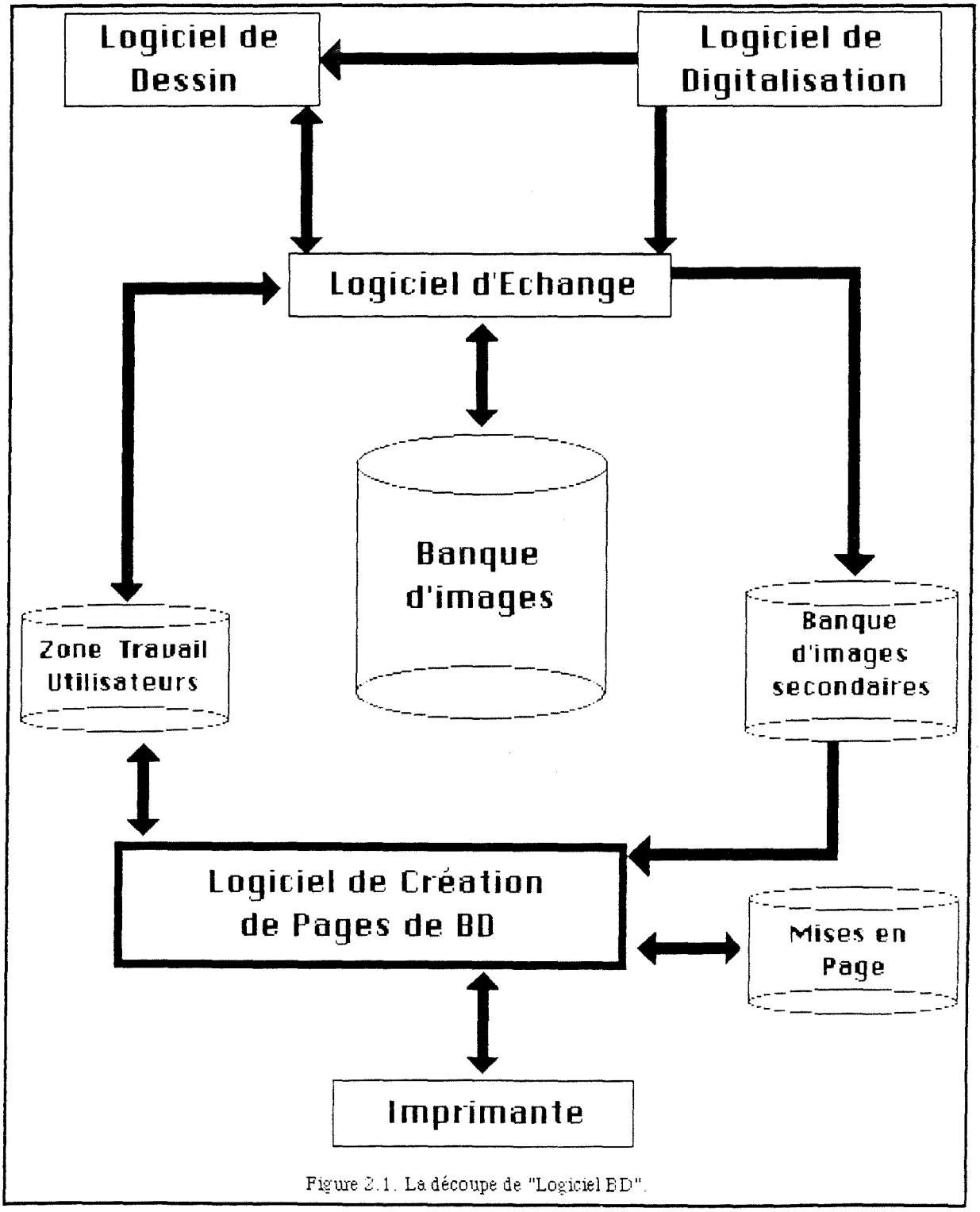


Figure 2.1. La découpe de "Logiciel BD".

* *Logiciels de dessin et de digitalisation*

"Logiciel BD" peut échanger des images avec d'autres logiciels graphiques via le Logiciel d'Echange. Remarquons qu'un logiciel de digitalisation peut directement fournir des images à un logiciel de dessin (pour y faire des retouches) sans passer par le logiciel d'échange. Les images provenant de l'extérieur (logiciel de dessin ou de digitalisation) sont soit rangées dans la B.I. Principale, soit dans la zone de travail d'un utilisateur. De plus, les images contenues dans la zone de travail d'un utilisateur ou dans la B.I principale peuvent être retravaillées par un logiciel de dessin.

Nous avons préféré faire appel à des logiciels existants sur le marché pour servir de source d'images. En effet, la réalisation d'un logiciel de dessin ou de digitalisation complet aurait largement dépassé le cadre de ce mémoire.

* *Le programme d'échange*

Comme nous l'avons vu, le programme d'échange est chargé "d'importer" des images provenant des sources extérieures dans la B.I. Principale et "d'exporter" des images provenant cette B.I. ou de la zone de travail d'un utilisateur vers des logiciels de dessin extérieurs. Ce logiciel d'échange s'occupe aussi de la gestion de l'organisation de la B.I. Principale[8] . C'est aussi grâce à lui que l'animateur peut créer ou mettre à jour les B.I. Secondaires.

* *La banque d'images principale*

La B.I. Principale consiste en une zone sur disquette ou, de préférence, sur disque dur, qui est organisée en vue de contenir des images

[8] c'est-à-dire, la création et la destruction des domaines de la B.I. ainsi que le transfert d'images entre les domaines de la B.I.

mises sous forme de fichiers. Cette B.I. ne peut être mise à jour[9] que par l'intermédiaire du programme d'échange d'images avec le monde extérieur[10]. Elle peut prendre une taille fort importante et il peut, par conséquent, être très difficile à un utilisateur mentalement déficient d'y sélectionner une image. C'est la raison pour laquelle nous avons décidé d'utiliser, lors de la création d'une page de B.D., une banque d'images de taille plus restreinte appelée **B.I. SECONDAIRE.**

* *La B.I. secondaire*

Il existe une et une seule B.I. Secondaire par utilisateur. Une B.I. Secondaire est en fait composée d'un nombre réduit d'images. Ces images ont été choisies dans la B.I. Principale, et uniquement dans celle-ci, par l'animateur en fonction du thème de la bande dessinée à réaliser ou bien des désirs de l'utilisateur mentalement déficient. Rappelons que c'est l'animateur et non l'utilisateur mentalement handicapé qui se charge de créer et de garnir la B.I. Secondaire ainsi que d'utiliser le programme d'échange.

Un utilisateur mentalement déficient ne peut créer une page de bandes dessinées que s'il possède une B.I. Secondaire qui lui est propre. C'est donc uniquement dans la B.I. Secondaire que le logiciel de création de B.D. va puiser les images dont il a besoin. L'utilisateur ne peut pas sélectionner directement une image dans la B.I. Principale.

* *La zone de travail des utilisateurs*

Il existe une et une seule zone de travail par utilisateur mentalement handicapé. Cette zone de travail est un emplacement sur disque dur ou sur disquette où chaque utilisateur place, quand il le désire, les pages de bandes dessinées qu'il a déjà réalisées ou qui sont en cours de réalisation ainsi que les dessins qu'il a sauvés ou qui proviennent du monde extérieur (via le programme d'échange). L'utilisateur pourra donc reprendre des pages de bandes dessinées qu'il a déjà créées pour les retravailler.

---

[9] recevoir de nouvelles images et changer sa structure.
[10] les programmes de dessin et de digitalisation.

L'ensemble des mises en page de l'utilisateur se retrouve également dans cette zone.

* *Le logiciel de création de page de BD*

Le logiciel de création de pages de bandes dessinées offre la possibilité à un utilisateur de créer une nouvelle page de bandes dessinées ou de continuer à travailler dans une page précédemment entamée.
Il propose une série d'outils qui permettent de travailler dans le dessin d'une case appartenant à une page de bandes dessinées. Parmi ces outils, on peut trouve la sélection d'images dans le B.I. Secondaire, l'agrandissement et la réduction d'images, le collage d'images dans le dessin de la case, le "crayonnage" au moyen de la souris, la création de bulles et de commentaires, la création de lignes, le découpage dans les images,...[11]

II.5. CONCLUSIONS

L'élaboration de l'énoncé final de "Logiciel BD" s'est faite à partir d'un énoncé de départ fort sommaire et d'une idée vague de ce que pouvait être ce type de logiciel. Un énoncé plus détaillé n'a pu être créé qu'après avoir analysé les caractéristiques de la population concernée par le logiciel et de ses exigences en ce qui concerne l'activité graphique au moyen d'ordinateurs personnels.

Il faut aussi savoir que nous ne disposons d'aucun modèle théorique de l'utilisateur mentalement déficient. Nous avons, en effet, montré[12] la difficulté de définir de manière rigoureuse les caractéristiques de cette population.

En plus des caractéristiques propres à cette population, il nous a fallu tenir compte des restrictions techniques des ordinateurs personnels existants sur le marché ainsi que des limites budgétaires du projet. Cela afin de concevoir un énoncé de projet qui puisse déboucher sur une réalisation concrète dans un délai raisonnable.

---

11 la liste complète des fonctionnalités apportées par ce logiciel est reprise au Chapitre III dans la partie concernant la structuration des traitements.
12 au chapitre I.

Chapitre III

La conception d'un système d'information

III.1. INTRODUCTION

Après avoir présenté, au chapitre II, l'énoncé final de "Logiciel BD", et afin de respecter les étapes créatives du cycle de vie du système d'information proposées par le Professeur F.Bodart co-auteur, avec Y.Pigneur, de l'ouvrage intitulé "Conception assistée des systèmes d'information" [BODPI, 89], nous avons décidé de décrire, dans ce chapitre, la "structuration des informations" ainsi qu'une partie de celle "des traitements". Nous avons donc utilisé le modèle "Schéma Entité/Association" pour la première structuration qui concerne les principales informations manipulées par le système. Ensuite, nous avons développé une partie du modèle de la "Structuration des traitements" en ne proposant que les applications et les phases du projet en question et en nous limitant à la simple définition des objectifs de ces dernières.

III.2. STRUCTURATION DES INFORMATIONS: LE SCHEMA ENTITE/ASSOCIATION

C'est à l'aide du schéma Entité-Association (E.A.) que nous exprimons la sémantique des données du système d'information (S.I.). Le schéma E.A. est une représentation graphique mettant en évidence la structuration des informations du S.I. Les différents concepts auxquels il est fait appel dans ce schéma sont largement définis dans l'ouvrage de F.Bodart et Y.Pigneur [BODPI, 89]. Nous en proposons cependant un résumé, nécessaire à la compréhension de la suite du texte:

- une entité est une chose concrète ou abstraite appartenant au réel perçu et à propos de laquelle on désire enregistrer des informations. Un type d'entité est une classe de toutes les entités possibles du réel perçu qui vérifient la définition constitutive du type.

- une association est définie par une correspondance entre deux ou plusieurs entités où chacune assume un rôle donné et à propos de laquelle on veut enregistrer des informations. Un type d'association est la classe de toutes les associations possibles du réel perçu qui vérifient la définition constitutive de type.

- un attribut est une caractéristique ou qualité d'une entité ou d'une association, et qui peut prendre plusieurs valeurs ou groupes de valeurs.

- une contrainte d'intégrité est une propriété, non représentée par les concepts définis ci-dessus et à laquelle doivent satisfaire les données appartenant à la mémoire du S.I.

III.2.1. Le schéma Entité/Association

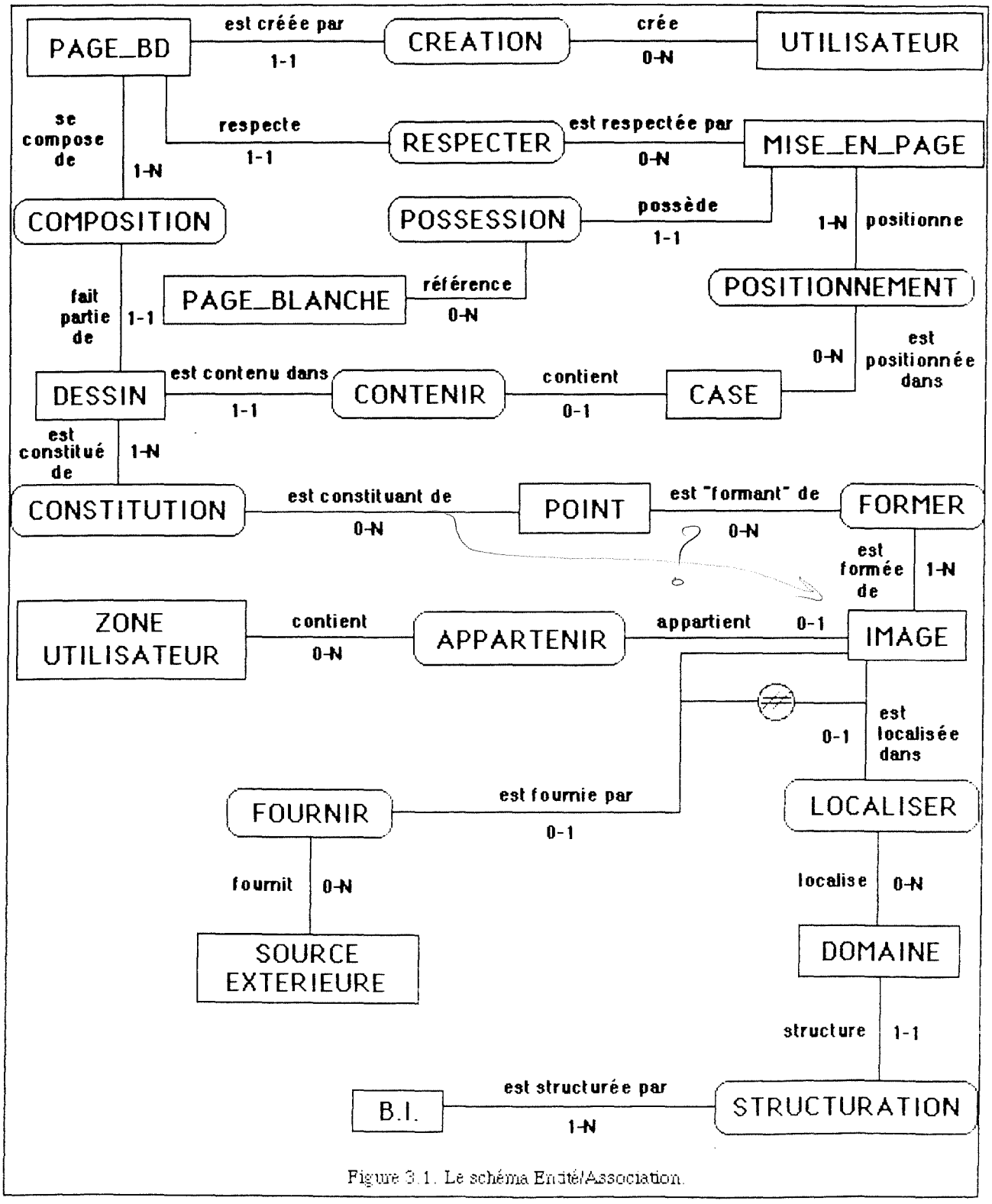


Figure 3.1. Le schéma Entité/Association.

III.2.2. Définition des éléments du schéma

III.2.2.1. Définition des entités

1. TYPE D'ENTITE: **PAGE_BD.**

DEFINITION: C'est une page de bandes dessinées réalisée ou en cours de réalisation.

IDENTIFIANT(S): num_page_bd + crée(UTILISATEUR)

ATTRIBUT(S): titre_page_bd

2. TYPE D'ENTITE: **CASE.**

DEFINITION: C'est un contour rectangulaire tracé sur la surface de la page et destiné à contenir un dessin.

IDENTIFIANT(S): dimension_case

ATTRIBUT(S): /

3. TYPE D'ENTITE: **UTILISATEUR.**

DEFINITION: Toute personne manipulant le logiciel de B.D. et désirant créer une page de bandes dessinées.

IDENTIFIANT(S): num_utilisateur

ATTRIBUT(S): nom_utilisateur
prénom_utilisateur
dnais_utilisateur
statut_utilisateur (animateur, handicapé)
sexe_utilisateur (M ou F)

4. TYPE D'ENTITE: **DESSIN.**

DEFINITION: Contenu graphique d'une case réalisé par assemblage d'images ainsi que des retouches éventuelles réalisées sur ces images.

IDENTIFIANT(S): num_dessin + se compose de(PAGE_BD)

ATTRIBUT(S): dimensions_dessin

5. TYPE D'ENTITE: **IMAGE.**

DEFINITION: représentation graphique d'une personne, d'une chose, ou d'un sujet quelconque.

IDENTIFIANT(S): num_image

ATTRIBUT(S): nom_image
dimensions_image
image_active (type booléen)

6. TYPE D'ENTITE: **SOURCE_EXTERIEURE.**

DEFINITION: lieu d'origine, de provenance d'images.

IDENTIFIANT(S): type_source (= digitaliseur, logiciel graphique)

ATTRIBUT(S): nom_logiciel
format_codage

7. TYPE D'ENTITE: **DOMAINE.**

DEFINITION: lieu de rassemblement d'images images concernant un même thème (par ex.:personnages, décors,...).

IDENTIFIANT(S): num_domaine.

ATTRIBUT(S): nom_domaine.

8. TYPE D'ENTITE: **B.I. (Banque d'Images).**

DEFINITION: lieu de rassemblement d'images rangées en fonction des domaines auxquels elles appartiennent.

IDENTIFIANT(S): nom_BI

ATTRIBUT(S): typeBI (BI_Principale / BI_Secondaire)

9. TYPE D'ENTITE: **POINT.**

DEFINITION: c'est la plus petite unité graphique constituant une image ou un dessin.

IDENTIFIANT(S): couleur_point

ATTRIBUT(S): /

10. TYPE D'ENTITE: **MISE_EN_PAGE.**

DEFINITION: disposition des différentes cases sur la page blanche d'un format déterminé ainsi que d'une orientation donnée.

IDENTIFIANT(S): num_mise_en_page

ATTRIBUT(S): /

11. TYPE D'ENTITE: **PAGE_BLANCHE.**

DEFINITION: Une face d'une feuille de papier.

IDENTIFIANT(S): format_page

ATTRIBUT(S): orientation_page

12. TYPE D'ENTITE: **ZONE UTILISATEUR.**

DEFINITION: C'est la zone en mémoire secondaire réservée à un seul utilisateur lui permettant de "stocker" les informations qui lui sont propres.

IDENTIFIANT(S): num_zone.

ATTRIBUT(S): nom_zone.

III.2.2.2. Définition des associations

1. TYPE D'ASSOCIATION: **CREATION.**

NOTATION: Création (crée : UTILISATEUR, est créée par: PAGE_BD)

DEFINITION: Une création représente l'association entre une page de bandes dessinées et le concepteur, créateur de cette page.

ASSOCIE: UTILISATEUR, PAGE_BD

ROLES:
Crée : UTILISATEUR
Est créée par : PAGE_BD

CONNECTIVITE:
0-N pour "crée" : UTILISATEUR
1-1 pour "est créée par" : PAGE_BD

IDENTIFIANT: (Utilisateur-crée, Page_bd-est-créée-par)

ATTRIBUTS: Date_création.

2. TYPE D'ASSOCIATION: **COMPOSITION.**

NOTATION: Composition (se compose de : PAGE_BD, fait partie de : DESSIN)

DEFINITION: Une composition associe à une page de bandes dessinées les dessins qui appartiennent à celle-ci.

ASSOCIE: PAGE_BD, DESSIN

ROLES:

Se compose de : PAGE_BD
Fait partie de : DESSIN

CONNECTIVITE:
1-N pour "se compose de" : PAGE_BD
1-1 pour "fait partie de" : DESSIN

IDENTIFIANT:
(Page_bd-se-compose-de, Dessin-fait-partie-de)

ATTRIBUTS: /

3. TYPE D'ASSOCIATION: **RESPECTER.**

NOTATION: Respecter (respecte : PAGE_BD, est respectée par: MISE_EN_PAGE)

DEFINITION: Respecter associe une mise en page à une page de bandes dessinées.

ASSOCIE: MISE_EN_PAGE, PAGE_BD

ROLES:
Respecte : PAGE_BD
Est respectée par : MISE_EN_PAGE

CONNECTIVITE:
0-N pour "est respectée par" : MISE_EN_PAGE
1-1 pour "respecte" : PAGE_BD

IDENTIFIANT: (Page_bd-respecte)

ATTRIBUTS: /

4. TYPE D'ASSOCIATION: **POSITIONNEMENT.**

NOTATION: Positionnement (positionne: MISE_EN_PAGE, est positionnée dans: CASE)

DEFINITION: Un positionnement crée un lien entre une case et sa position dans une mise en page.

ASSOCIE: MISE_EN_PAGE, CASE

ROLES:
Positionne: MISE_EN_PAGE
Est positionnée dans : CASE

CONNECTIVITE:
1-N pour "positionne" : MISE_EN_PAGE
0-N pour "est positionnée dans" : CASE

IDENTIFIANT:
(Mise_en_page-positionne, Case-est-positionnée-dans)

ATTRIBUTS: /

5. TYPE D'ASSOCIATION: **CONTENIR.**

NOTATION: Contenir (contient : CASE, est contenu dans: DESSIN)

DEFINITION: Contenir représente une association entre une case et un dessin.

ASSOCIE: CASE, DESSIN

ROLES:
Est contenu dans: DESSIN

Contient: CASE

CONNECTIVITE:
0-1 pour "contient" : CASE
1-1 pour "est contenu dans" : DESSIN

IDENTIFIANT: (Dessin-est-contenu-dans)

ATTRIBUTS: /.

6. TYPE D'ASSOCIATION: **CONSTITUTION.**

NOTATION: Constitution (est constitué de : DESSIN, est constituant de: POINT)

DEFINITION: Constitution représente la localisation des points qui constituent le dessin.

ASSOCIE: DESSIN, POINT

ROLES:
Est constitué de : DESSIN
Est constituant de : POINT

CONNECTIVITE:
0-N pour "est constituant de" : POINT
1-N pour "est constitué de" : DESSIN

IDENTIFIANT:
(Dessin-est-constitué-de, Point-est-constituant-de)

ATTRIBUTS: Position_du_point_dans_le_dessin.

7. TYPE D'ASSOCIATION: **FORMER.**

NOTATION:
Former (est "formant" de: POINT, est formée de: IMAGE)

DEFINITION: Former représente la position des points qui forment l'image.

ASSOCIE: POINT, IMAGE

ROLES:
Est "formant" de: POINT
Est formée de: IMAGE

CONNECTIVITE:
0-N pour "est 'formant' de": POINT
1-N pour "est formée de": IMAGE

IDENTIFIANT: (Image-est-formée-de, Point-est-"formant"-de)

ATTRIBUTS: Position_du_point_dans_image.

8. TYPE D'ASSOCIATION: **FOURNIR.**

NOTATION: Fournir (est fournie par : IMAGE, fournit: SOURCE_EXTERIEURE)

DEFINITION: Fournir nous dit si une image est fournie par un logiciel graphique ou par un digitaliseur.

ASSOCIE: IMAGE, SOURCE_EXTERIEURE

ROLES:
Est fournie par: IMAGE
Fournit : SOURCE_EXTERIEURE

CONNECTIVITE:
0-N pour "fournit": SOURCE_EXTERIEURE
0-1 pour "est fournie par": IMAGE

IDENTIFIANT:
(Image-est-fournie-par, Source_extérieure-fournit)

ATTRIBUTS: /.

9. TYPE D'ASSOCIATION: **LOCALISER.**

NOTATION: Localiser (est localisée dans : IMAGE, localise : DOMAINE)

DEFINITION: Localiser représente le lien d'appartenance entre une image et un domaine.

ASSOCIE: IMAGE, DOMAINE

ROLES:
Appartient : IMAGE
Contient : DOMAINE

CONNECTIVITE:
0-N pour "localise" : DOMAINE
0-1 pour "est localisée dans" : IMAGE

IDENTIFIANT: (Image-est-localisée-dans, Domaine-localise)

ATTRIBUTS: /.

10. TYPE D'ASSOCIATION: **STRUCTURATION.**

NOTATION:
Structuration (structure : DOMAINE, est structurée de : B.I.)

DEFINITION: Structuration représente la localisation, c'est-à-dire le nom du chemin, d'un domaine appartenant à la banque d'images.

ASSOCIE: DOMAINE, B.I.

ROLES:
Structure : DOMAINE
Est structurée par : B.I.

CONNECTIVITE:
1-N pour "est-structurée-par" : B.I.
1-1 pour "structure" : DOMAINE

IDENTIFIANT: (Domaine-structure, B.I.-est-structurée-par)

ATTRIBUTS: Nom_chemin_du_domaine.

11. TYPE D'ASSOCIATION: **POSSESSION.**

NOTATION:
Possession (possède : MISE_EN_PAGE, référence : PAGE_BLANCHE)

DEFINITION: Possession représente le lien entre une mise-en-page et sa page blanche de base.

ASSOCIE: MISE_EN_PAGE, PAGE_BLANCHE.

ROLES:
Possède : MISE_EN_PAGE

Référence : PAGE_BLANCHE.

CONNECTIVITE:
0-N pour "référence" : PAGE_BLANCHE
1-1 pour "possède" : MISE_EN_PAGE.

IDENTIFIANT:
(Page_blanche-référence, Mise_en_page-possède).

ATTRIBUTS: /

12. TYPE D'ASSOCIATION: **APPARTENIR.**

NOTATION: Appartenir (appartient : IMAGE, contient : ZONE UTILISATEUR)

DEFINITION: Appartenir donne le lien d'existence entre une image et une zone de travail d'un utilisateur.

ASSOCIE: IMAGE, ZONE UTILISATEUR

ROLES:
Appartient : IMAGE
Contient : ZONE UTILISATEUR

CONNECTIVITE:
0-N pour "contient" : ZONE UTILISATEUR
0-1 pour "appartient" : IMAGE

IDENTIFIANT: (Zone utlisateur-contient, Image-appartient)

ATTRIBUTS: /

III.2.2.3. Contrainte

- Seules les images de la B.I. appartiennent à un domaine. C'est pourquoi il existe une contrainte d'exclusion entre les rôles "Image-est-localisée-dans" et "Image-est-fournie-par". En effet, une image ne peut remplir ces deux rôles en même temps.

III.3. STRUCTURATION DES TRAITEMENTS

Nous proposons ici une structuration des traitements effectuée jusqu'au niveau des phases. Il s'agit en fait de découper le projet par décomposition en plusieurs niveaux de traitements globaux en une série de traitements plus élémentaires. Le résultat obtenu est une arborescence des traitements.

III.3.1. Projet: "LOGICIEL DE BANDES DESSINEES"

Le terme projet est utilisé pour définir la partie du système d'information qui fait l'objet de l'analyse. Il s'agit, dans le cas qui nous préocuppe, du logiciel de création de pages de bandes dessinées pour personnes mentalement déficientes ("Logiciel BD").

III.3.2. Applications

Le terme application sert à décrire un traitement quasi-autonome par rapport aux autres applications: elle constitue une unité de planning dans la gestion d'un projet. C'est la plus petite partie d'un projet dont le cycle de développement doit être considéré globalement. Un premier niveau de décomposition propose la découpe du projet "Logiciel BD" en trois applications intitulées "Préparation Page BD", "Interaction Environnement Extérieur" et "Gestion Banque d'Images".

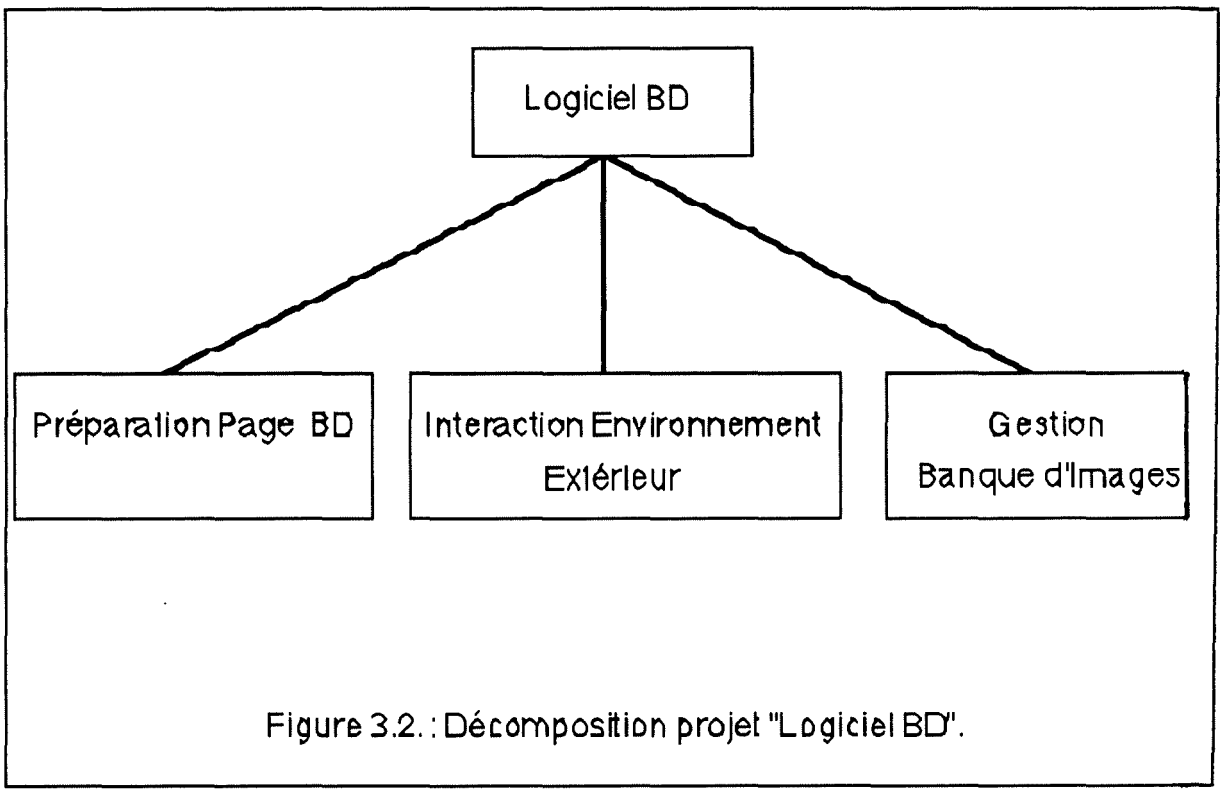


Figure 3.2. : Décomposition projet "Logiciel BD".

Application: **PREPARATION PAGE BD**
Objectif: Concerne le travail dans une page de bandes dessinées en commençant par l'élaboration d'une mise-en-page puis en travaillant dans chaque case de cette page.

Application: **GESTION DE LA BANQUE D'IMAGES (B.I.).**
Objectif: Concerne les opérations de mises à jour et de consultation réalisées sur la B.I..

Application: **INTERACTION ENVIRONNEMENT EXTERIEUR.**
Objectif: Concerne l'échange d'images avec l'extérieur (autres logiciels de graphisme, digitaliseurs,...).

III.3.3. Phases

Chaque application peut être décomposée en traitements plus fins. Ces derniers sont appelés des phases dans la terminologie de [BODPI, 89]. Une phase est un traitement possédant une unité spacio-temporelle d'exécution. Dans notre cas, chaque phase correspond à une fonctionnalité de l'énoncé défini au chapitre II.

III.3.3.1. Application: PREPARATION PAGE BD

Phase: **Coloriage.**
Objectif: Permet le coloriage à main levée avec le crayon dans l'image sélectionnée (image active) ou dans le dessin de la case selon une couleur et une épaisseur de trait (pour le crayon) choisies.

Phase: **Gommage.**
Objectif: Proposer un choix de tailles de gommes et effacer la partie se situant sous le curseur de la souris dans l'image active (ou dans le dessin).

Phase: **Pot_De_Peinture.**
Objectif: Permettre de choisir une couleur de remplissage dans la palette de couleurs. Puis, remplir une zone se trouvant dans l'image active ou dans le dessin de la case avec la couleur choisie.

Phase: **Découpage.**
Objectif: Rendre active une partie d'image découpée dans le dessin de la case actuelle de travail, au moyen de la souris, en traçant le contour de la zone dans laquelle on veut extraire l'image.

Phase: **Variation_Taille_Image_Active.**
Objectif: Faire varier la taille de l'image active en l'agrandissant ou en la diminuant dans des proportions voulues.

Phase: **Rotation_Image_Active.**
Objectif: Faire pivoter l'image active dans le plan de l'écran en tenant compte du sens de rotation ainsi que de l'angle de rotation désirés par l'utilisateur.

Phase: **Sélection_Image.**
Objectif: Permettre à l'utilisateur de sélectionner une image dans la B.I. secondaire afin de travailler avec elle dans la page BD. L'image choisie deviendra la nouvelle image active.

Phase: **Création_Bulle_Dialogue.**
Objectif: Permettre d'introduire le texte du dialogue au moyen du clavier et de choisir un type de caractère. Ensuite, une bulle de dialogue est créée et affichée dans le dessin.

Phase: **Création_Commentaire.**
Objectif: Permettre d'introduire le texte du commentaire au clavier et de choisir un type de caractère. Ensuite, un commentaire est créé et affiché dans le dessin.

Phase: **Affichage_Page_Travail.**
Objectif: Afficher l'entièreté de la page de BD dans son état actuel d'avancement. Indiquer dans cette page la case actuelle de travail et permettre de choisir une autre case de travail actuelle.

Phase: **Sélection_Mise_En_Page.**
Objectif: Proposer l'ensemble des mises en page qui ont déjà été créées et sauvées. Permettre de choisir une mise en page parmi cet ensemble. La mise en page choisie sera celle de la page actuelle de travail.

Phase: **Création_Mise_En_Page.**
Objectif: Proposer l'ensemble de formats de pages disponibles et donner la possiblité d'en choisir un parmi eux. Ensuite, permettre de créer des cases de taille voulue par étirement de rectangles avec la souris et de placer ensuite ces cases à l'endroit voulu sur la page. La page apparaîtra entièrement à l'écran avec les cases déjà réalisées ainsi que celles que l'on construit.

Phase: **Sauvetage_Mise_En_Page_Créée.**
Objectif: Sauver dans la zone de travail de l'utilisateur la mise en page qui vient d'être créée sous un nom donné.

Phase: **Chargement_Page_BD_Sauvée.**
Objectif: Proposer les noms de toutes les pages de BD contenues dans la zone de travail de l'utilisateur. Permettre d'en choisir une et aller la chercher dans la zone de travail de l'utilisateur pour en faire la page actuelle de travail.

Phase: **Sauvetage_Page_BD.**
Objectif: Permettre l'introduction d'un nom pour la page BD actuelle. Sauver dans la zone de travail de l'utilisateur la page BD actuelle sous le nom introduit.

Phase: **Impression_Page_BD.**
Objectif: Afficher à l'écran la page de travail en entier et préparer l'envoi à l'imprimante, dans les meilleures conditions, de cette page de B.D..

Phase: **Impression_Dessin_Case_Actuelle.**
Objectif: Proposer le choix d'un dessin et l'envoyer vers l'imprimante.

III.3.3.2. Application: INTERACTION ENVIRONNEMENT EXTERIEUR

Phase: **Importation_Image.**
Objectif: Demande le chemin donnant accès à ou aux image(s) à "importer" ainsi que le type de l'image à "importer". Affiche l'ensemble des images susceptibles d'être "importées" se trouvant à l'endroit spécifié par le chemin. Permet le choix de l'image ou des images à "importer". Convertit la ou les image(s) sélectionnée(s) dans le format de travail de "Logiciel BD" et place ces images converties dans la zone de travail de l'utlisateur ou dans la B.I. Principale suivant le choix.

Phase: **Exportation_Image.**
Objectif: Afficher l'ensemble des images se trouvant dans la zone de travail de l'utilisateur. Permettre la sélection de celle(s) que l'on veut convertir. Convertir la ou les image(s) sélectionnée(s) dans un des formats disponibles et placer cette ou ces image(s) dans la zone déterminée par un chemin donné.

III.3.3.3. Application: GESTION BANQUE D'IMAGES

Phase: **Ajout_Image_BI.**
Objectif: Afficher l'ensemble des images de la zone de travail de l'utilisateur, qui sont susceptibles d'être introduites dans la B.I.. Permettre la sélection de l'image que l'on veut ajouter à la B.I.. Ajouter l'image sélectionnée dans le domaine spécifié par un chemin de la B.I. après lui avoir donné un nom.

Phase: **Destruction_Image_BI.**
Objectif: Permettre d'introduire un chemin dans la B.I.. Afficher l'ensemble des images contenues dans le domaine BI déterminé par le chemin_BI donné. Permettre la sélection de l'image que l'on veut supprimer dans la B.I.. Supprimer de la B.I. l'image sélectionnée.

Phase: **Création_Domaine_BI.**
Objectif: Permettre de déterminer un chemin de la B.I.. Afficher le ou les sous-domaine(s) contenu(s) dans le domaine_BI spécifié par le chemin_BI introduit. Créer un nouveau sous-domaine de nom donné dans ce domaine.

Phase: **Suppression_Domaine_BI.**
Objectif: Permettre de choisir un chemin de la B.I.. Afficher le ou les sous-domaine(s) contenu(s) dans le domaine de la B.I. spécifié par le chemin introduit. Permettre la sélection du domaine à éliminer. Supprimer de la B.I. le domaine sélectionné s'il est vide.

Phase: **Garnir_BI_Secondaire.**
Objectif: Afficher par domaine l'ensemble des images de la B.I.. Aller chercher les images sélectionnées dans les domaines BI déterminés par les chemins_BI donnés pour les placer dans la BI secondaire. Cet ensemble d'images sélectionnées sera disponible pour le travail futur dans une page de bandes dessinées.

III.4. CONCLUSIONS

Nous venons de déterminer, en réalisant les structuration partielles des informations et des traitements, quelles sont les principales informations manipulées par le logiciel ainsi que les grandes fonctionnalités que "Logiciel BD" doit respecter. Cette analyse fonctionnelle partielle a servi de base à la découpe orientée objet qui fait l'objet des chapitres suivants. Nous avons ainsi décomposé notre logiciel en trois grandes parties[1]. C'est aussi au cours de cette analyse que nous avons, en fait, pu fixer avec précision les fonctionalités du "Logiciel BD" en collaboration avec les principaux intéressés c'est-à-dire, les animateurs d'ateliers pour personnes handicapées mentales.

[1] les applications "Préparation Page BD", "Gestion Banque d'Images" et "Environnement extérieur".

Chapitre IV

La théorie orientée objet

Préliminaire: Le terme "objet" est, dans la littérature courante relative à la théorie orientée objet, employé comme synonyme soit de classe, soit d'instance (cfr.infra). Nous avons décidé, arbitrairement et logiquement, afin d'éviter toute ambiguïté, de conserver au terme "objet" son sens de tous les jours et de parler en termes de "classe d'objet" et d'"instance d'objet".

IV.1. INTRODUCTION

Le "Logiciel BD" ayant été conçu en s'inspirant de la théorie orientée objet, il nous paraît opportun de présenter, dans un tout premier temps, cette approche au lecteur. La plupart des idées et des concepts qu'elle englobe ne sont cependant pas nouveaux dans le domaine du développement de logiciels.

Avant de présenter les principes fondamentaux de cette théorie, nous avons jugé bon de présenter les objectifs poursuivis par toute méthodologie de conception de logiciels et, en particulier par celle qui

nous préocuppe. Vient finalement une présentation des mécanismes qui tentent de mettre en oeuvre cette théorie. Nous avons séparé, dans ce chapitre, l'approche théorique de l'approche plus technique concernant les mécanismes orientés objet car ceux ci ne sont qu'une manière de mettre en oeuvre, de façon souvent incomplète, l'idée qui se trouve derrière la théorie.

IV.2. PHILOSOSPHIE ET PRINCIPES DE LA THEORIE ORIENTEE OBJET

Pour bien comprendre la philosophie et les principes de la théorie orientée objet, il faut au préalable connaître les objectifs poursuivis par tout méthodologie de développement de logiciels. Ensuite, il est important de voir de quelle façon la théorie orientée objet tente d'atteindre ces buts; manière d'atteindre les objectifs qui correspond aux fondements de la théorie.

IV.2.1. Buts de toute méthodologie de développement de logiciels

Un des buts principaux de l'ingéniérie logicielle, "Software Engineering", est d'aider les concepteurs de logiciels à produire des programmes de qualité. Les notions relatives à cette qualité ont notamment été abordées au cours de "Méthodologie de développement de logiciels" [AVLAM,90] de seconde licence.

Nous présentons ici les qualités qu'un software devrait respecter, dégagées par Meyer dans son ouvrage "Object Oriented Software Construction" [MEYER, 88] et qui sont au nombre de dix.

1. Un logiciel doit être "correct" (CORRECTNESS). C'est-à-dire qu'un logiciel doit effectuer de façon précise sa tâche telle qu'elle a été définie dans le cahier des charges.

2. Un logiciel doit être "robuste" (ROBUSTNESS). Il doit pour cela pouvoir fonctionner, même dans des situations anormales.

3. Un logiciel doit être "extensible" (EXTENDIBILITY). L'extensibilité d'un logiciel est la facilité avec laquelle il peut être adapté aux changements de spécifications.

4. Un logiciel doit être "réutilisable" (REUSABILITY). La réutilisabilité est la capacité qu'un programme a à être réutilisé, dans son entièreté ou en partie, pour de nouvelles applications.

5. Un logiciel doit être "compatible" (COMPATIBILITY). La compatibilité est la facilité avec laquelle un logiciel peut être combiné avec d'autres.

6. Un logiciel doit être "efficace" (EFFICIENCY). Il doit, pour cela, faire bon usage des ressources "hardware", telles que les processeurs, les mémoires internes et externes et les appareils de communication.

7. Un logiciel doit être "portable" (PORTABILITY). La portabilité est l'aisance avec laquelle un programme peut être transféré vers différentes machines ou vers d'autres environnements.

8. Un logiciel doit être "vérifiable" (VERIFIABILITY). Il doit ainsi être aisé de préparer des procédures de tests pour détecter des anomalies et de les suivre durant les phases de tests du logiciel.

9. Un logiciel doit être "consistant" (INTEGRITY). La consistance est la capacité qu'ont les logiciels à protéger leurs divers composants des accès et modifications extérieures non-autorisées.

10. Un logiciel doit être "facile à utiliser" (EASY TO USE). Il doit être aisé d'apprendre à se servir du logiciel, de préparer les données à entrer, d'analyser les résultats et de remédier aux erreurs de manipulation.

IV.2.2. Comment atteindre ces objectifs?

Nous allons nous intéresser, dans ce point, à la manière d'atteindre certains de ces objectifs. Ces objectifs ne peuvent cependant pas être tous maximisés en même temps. En effet, le fait de favoriser certains d'entre-eux peut en désavantager d'autres. Ce sont alors les objectifs d'extensibilité et réutilisabilité qui seront considérés comme les plus importants.

Nous présentons, pour commencer, la solution apportée par la modularité[1]. Vient ensuite celle apportée par la théorie orientée objet qui n'est qu'une autre façon de modulariser.

IV.2.2.1. Modularité

La **modularité** permet de réaliser les objectifs de "réutilisabilité" et "d'extensibilité" d'un logiciel. Cette notion n'est pas nouvelle et n'est pas propre à la philosophie orientée objet. De plus, la modularité s'applique aussi bien à la programmation et aux spécifications qu'à la conception même du logiciel.

C'est parce qu'on constate que la programmation et la conception sont des tâches hautement répétitives où l'on utilise fréquemment les mêmes structures et les mêmes actions qu'il devient nécessaire de réutiliser ce qui a déjà été réalisé.

La modularité consiste à voir la construction d'un programme comme étant l'assemblage de différentes pièces (les modules). Les modules sont autonomes, cohérents et organisés dans des architectures robustes. "Ces modules possèdent différentes caractéristiques et doivent respecter certains principes" [AVLAM,90]. Le faible degré de couplage et le masquage de l'information sont les plus importants parmi ces principes. Le faible degré de couplage signifie que "si deux modules communiquent ensemble, il doivent échanger aussi peu d'information que possible" [MEYER, 88]. Par masquage de l'information, on entend que "toute l'information concernant un module devrait être privée à celui-ci à moins qu'elle n'ait été déclarée *publique"* [MEYER, 88].

A un plus haut niveau de structuration des programmes, on peut considérer les modules comme étant des "paquets" (packages) [MEYER, 88] qui peuvent rassembler plus d'une routine avec des déclarations de types, de constantes et de variables. Ce type de modules offre une meilleure technique d'encapsulation que les routines puisqu'ils réunissent les structures de données et les opérations qui y sont associées. Des langages de programmation modulaire, tels que Ada ou Modula-2, permettent de mettre en oeuvre de tels concepts.

---

[1] ce concept n'est pas nouveau pour nous.

Il existe deux techniques qui étendent la flexibilité de l'utilisation de ces "paquets". La première est l'**overloading.** Elle consiste en l'utilisation d'un même nom pour plus d'une opération, ce qui revient à attacher plus d'une signification à un même nom apparaissant dans un programme. Ceci implique que nous aurons affaire à des procédures différentes mais ayant le même nom. Remarquons toutefois que l'overloading est essentiellement une facilité syntaxique. La seconde technique est la **généricité** qui est "la capacité de définir des modules paramétrés" [MEYER, 88]. De tels modules, appelés modules génériques, ne sont pas utilisables directement. Il s'agit simplement de schémas de modules dont les paramètres sont des types. Les véritables modules, appelés instances de modules génériques, sont obtenus en fournissant des types effectifs pour chacun des paramètres génériques. Ces deux techniques ne résolvent cependant pas tous les problèmes de réutilisablité. "Ce qui est nécessaire, c'est de "capturer" les parties communes entre les groupes de structures de données liées, et des techniques pour isoler les clients de la représentation interne des modules." [MEYER, 88]

IV.2.2.2. La vision orientée objet

La conception **orientée objet** base la structure des systèmes sur les classes des objets qu'ils manipulent. En orienté objet, l'entité principale est la **donnée** au contraire de la programmation conventionnelle où l'élément de base est la procédure. Meyer donne une définition intéressante de la conception orientée objet: "La conception orientée objet est la méthode qui conduit à des architectures logicielles basées sur les objets que manipule chaque système ou sous-système (plutôt que sur la fonction qu'il est sensé assurer)" [MEYER, 88].

Une donnée est considérée soit comme une valeur abstraite et immatérielle telle qu'un nombre ou une liste, soit comme un objet concret et matériel tel qu'une variable ou un "mot mémoire". "C'est cette deuxième optique qui est prise en compte en programmation orientée objet. Une donnée est un objet, vu comme un objet au sens de la vie courante, que l'on peut construire, transformer, déplacer, détruire, etc..." [LECHA,91]. En conséquence de quoi, deux variables d'un type quelconque seront en fait la donnée elle-même et donc toujours différentes en tant qu'objets. Au

contraire de l'autre optique où "une variable n'est pas une donnée mais seulement une "mémoire" contenant la donnée ou, plus exactement, une représentation de celle-ci" [LECHA,91].

Pourquoi utiliser les données comme "clé" de la modularisation des systèmes?

Selon Meyer [MEYER, 88], cette façon de procéder permet d'atteindre les buts de "compatibilité", de "réutilisabilité" et d'"extensibilité".
La "compatibilité" est une des principales motivations qui font que l'on utilise les données comme critère de décomposition. "Il est difficile de combiner des actions si les données auxquelles elles accèdent ne sont pas prises en considération" [MEYER, 88].
La "réutilisabilité" possède aussi des arguments. "Pour toutes les applications qui englobent des structures de données non-triviales, il est difficile de construire des composants réutilisables s'ils mettent en forme seulement des actions et ignorent les données" [MEYER, 88].
Mais c'est la "continuité" qui apporte l'argument le plus convainquant. "Les structures de données, du moins si elles sont vues à un niveau d'abstraction suffisant, sont les éléments du système les plus stables dans le temps" [MEYER, 88].

Les types de données abstraits

Etant donné que la programmation orientée objet se revendique de la théorie mathématique des types abstraits, nous commencons par décrire le concept de **Types de Données Abstraits.**

Un **type de données** décrit un ensemble d'objets qui ont la même représentation, que ce soit un type simple comme les entiers ou les réels, ou un type complexe comme les tableaux. Il existe également un certain nombre d'opérations associées à chaque type de données comme, par exemples, effectuer un calcul arithmétique avec des entiers, ou concaténer deux "strings" de caractères.

Les **types de données abstraits,** quant à eux, étendent la notion de "type de données" en cachant l'implémentation des opérations (définies

par le programmeur) qui lui sont associées. Comme nous l'avons vu plus haut, cette possibilité de masquer l'information rend possible le développement des composants d'un logiciel pour qu'ils soient facilement réutilisables et extensibles. Ceci engendre l'apparition d'une séparation entre, d'un côté, l'interface du type de données et, d'un autre côté, son implémentation. Cette implémentation se compose de la représentation[2] du type et des opérations que l'on peut effectuer dessus. Ainsi, la représentation interne et/ou l'implémentation d'une opération peuvent être modifiées, à tout moment, sans que l'interface du type de données abstrait en soit affectée. Donc, que nous décidions, par exemple, d'utiliser un tableau ou une liste afin de ranger des informations, ne devrait normalement causer aucune modification dans l'interface publique.

Un langage qui supporte la notion de "types de données abstraits", va donner la possibilité de définir, pour un type de données, une structure de données et des opérations qui lui seront associées. Ces fonctions seront utilisées afin de manipuler les occurrences du type défini par la structure de données. De plus, toutes les manipulations des instances d'objet (du type de données) se feront exclusivement à travers ces opérations.

Meyer donne alors une seconde définition de la conception orientée objet basée sur les types de données abstraits: "La conception orientée objet est la construction de systèmes logiciels en tant que collections structurées d'implémentations de types de données abstraits" [MEYER,88].

Les classes d'objet

Le concepteur d'une application peut choisir de définir certains objets en termes de structure de données et d'opérations, différemment par rapport à d'autres, tandis que certains autres peuvent avoir un comportement tout à fait similaire. Ceux qui partagent le même comportement appartiennent à un même ensemble appelé **classe d'objet** (ou *type* d'objet). Celle-ci est définie comme une spécification générique d'un nombre arbitraire d'objets similaires. Nous pouvons donc considérer une classe comme étant un gabarit pour un type spécifique d'éléments.

---

[2] c'est-à-dire une structure de données.

Elle permet ainsi de construire la "taxonomie" des occurences d'objets à des niveaux d'abstraction ou conceptuels différents. En d'autres mots, les classes d'objet permettent de décrire, à un endroit donné, les comportements génériques d'un ensemble d'objets.

Les différents types de classes

* *Les Sous-Classes*

Une **sous-classe** est une classe d'objet qui hérite du comportement et/ou de la structure des données d'une autre classe. Elle ajoute d'habitude ses propres comportements et structure afin de se spécialiser.

* *Les Super-Classes*

Une **super-classe** est une classe d'objet de laquelle un comportement et/ou la structure des données spécifique est (sont) hérité(s).

* *Les Classes Abstraites*

Toutes les classes d'objet ne créent pas nécessairement des instances d'elles-mêmes. Nous appelons alors **classes abstraites** (par opposition aux classes concrètes) les classes qui n'existent pas pour produire des instances d'elles-mêmes. Elles existent pour que le comportement commun à une variété de classes puisse être extrait et placé en un endroit accessible par chacune, où il pourra être défini une fois pour toutes, et réutilisé maintes fois.

* *Les Métaclasses*

Les **métaclasses** sont des classes d'objet dont les instances sont également des classes. Tout comme la classe décrit les variables à instancier et les méthodes applicables à ses instances, la métaclasse décrit les variables et le comportement de la classe considérée comme une occurence. Cependant, tous les langages orientés objet ne supporte pas nécessairement ce type de classe.

Les instances d'objet

Les objets qui se comportent de la manière spécifiée par une classe sont appelés **instances d'objet,** ou occurrences de cette classe. Chaque occurrence d'objet (du système) est l'instance d'une et une seule classe particulière. Lorsque une occurrence est créée, elle se comporte comme toutes les autres appartenant à la même classe d'objet.

Les variables à instancier

Les **variables à instancier,** également appelées attributs, d'une classe d'objet définissent la spécificité d'une instance particulière. Elles reçoivent une valeur lors d'une instanciation. Les valeurs de ces attributs fournissent l'état des instances d'objet de la classe considérée.

IV.3. LES MECANISMES DE FONCTIONNEMENT

Les mécanismes de fonctionnement sont les règles qui sont à la base du principe de fonctionnement de la conception et de la programmation orientées objet. Ceux-ci mettent en oeuvre les notions d'**encapsulation,** de **masquage de l'information, de polymorphisme** (vus plus haut) ainsi que la notion d'**héritage.** Ces quatre concepts jouent un rôle très important dans la théorie orientée objet.

Nous donnons, dans les points qui suivent, la façon dont certains de ces mécanismes sont implémentés dans le langage C++ qui est une version améliorée du langage C développé par les Laboratoires de Bell. Il était initialement appelé "C with classes". C++ a été développé, à l'origine, pour faciliter la gestion, la programmation, et maintenir de gros systèmes de software. Il a ensuite évolué pour rencontrer les buts suivants:
* conserver une grande efficacité au niveau machine et la portabilité pour lesquelles C était réputé;
* conserver la compatibilité entre C et C++;
* améliorer le C avec les nouveaux principes dont celui de "masquage de l'information".

IV.3.1. L'encapsulation

Comme nous avons déjà vu dans les points précédents, l'encapsulation consiste à rassembler, dans une même boîte, certaines connaissances et opérations qui ont une relation conceptuelle commune. L'encapsulation est très importante dans l'approche orientée objet. "Elle contribue à restreindre les effets dûs aux changements en plaçant un mur de code autour de chaque morceau de données." [COX,86].

Une des plus importantes caractéristiques de C++, que l'on ne rencontre pas dans le C normal, est le concept de classe. Il est possible de définir de nouveaux types de données au moyen de la déclaration *class*. C'est en fait une généralisation de l'ancienne déclaration *struct* du C. *Class* permet la création d'un type de conglomérat défini par le programmeur. Dans sa définition, nous pouvons inclure de l'information à propos de variables et aussi définir précisément quelles fonctions ont accès aux variables. Les fonctions définies dans une classe sont appelées fonctions membres *(member functions)*. Ceci signifie qu'elles peuvent agir sur n'importe quelle variable définie dans cette classe.

De nombreuses classes nécessitent une étape d'initialisation lors du lancement et une étape de destruction lorsque ces classes ne sont plus nécessaires. En ce qui concerne la première étape, C++ met en oeuvre un mécanisme qui produit automatiquement l'**initialisation** nécessaire chaque fois qu'une variable de cette classe est créée, et cela sans devoir faire un appel explicite à une fonction d'initialisation. C++ appelle cette fonction, fonction constructrice *("constructor function")*. Quant à la seconde étape, elle concerne l'opération de **désinitialisation** souvent nécessaire d'une classe. On appellera les fonctions qui se chargent de cette opération, fonctions destructrices *("destructor function")*. Une de leurs principales utilités est de désallouer, en mémoire, ce qui a été alloué par la fonction constructrice. La fonction de destruction est utilisée lorsqu'une variable de type *class* n'existe plus. Ceci se produit lorsque sa portée est terminée ou lorsque le programme se termine.

IV.3.2. Le masquage de l'information ("Information-hiding")

Si une classe d'objet possède, de façon bien distincte, une interface publique et une représentation privée, telles qu'elles ont été définies antérieurement, alors elle respecte le principe du **masquage de l'information** appelé en anglais *Information-hiding*. Chaque classe est donc composée d'un aspect public qui sert d'interface avec les autres entités du système. Cette zone publique définit les opérations publiquement disponibles, tant du point de vue des attributs qui peuvent être manipulés que de celui des informations qui peuvent être fournies. En d'autres termes, cette interface décrit toutes les interactions possibles entre l'objet et le monde extérieur. Chaque classe d'objet possède aussi un aspect privé. Ce dernier représente la structure de la classe et la manière selon laquelle les méthodes sont implémentées. Car la façon dont une instance d'objet effectuera ses opérations ou calculera l'information ne concerne en aucune manière les autres parties du système. En respectant ce processus, les classes sont libres de modifier leur aspect privé que ce soit en changeant leur représentation interne ou en implémentant un nouvel algorithme, sans pour autant affecter le reste du système. Le masquage de l'information permet ainsi de soustraire à la vue certaines choses qui ont été encapsulées dans l'objet. Celui-ci ne révèle alors publiquement que ses capacités (ce qu'il peut faire) à savoir le nom de ses fonctions, et non la façon dont il "agit".

En C++, le mot-clé *public* contrôle l'accès aux éléments de la classe. Les identifiants qui sont déclarés avant le mot public peuvent être accédés seulement par les fonctions qui sont membres de la classe. Ils sont appelés identifiants privés *("private identifiers")*. Les identifiants que l'on retrouve après public sont accessibles par n'importe quelle fonction du programme. C++ tente ainsi de conserver la liberté et l'esprit de C en permettant de créer des **unités fonctionnelles,** ou boîtes noires, qui ont un accès strictement contrôlé.

Il est possible de donner, à une ou plusieurs fonctions, l'accès à la partie privée d'une classe, et cela sans pour autant les rendre membres de la classe. Pour ce faire, il faut faire précéder leur déclaration dans la classe par le mot-clé *friend*. Ces fonctions amies *friend functions* sont des fonctions C conventionnelles. Elles n'ont pas de lien particulier avec les

classes, excepté le fait qu'elles peuvent référencer n'importe quel membre de données qui ont été déclarées privées.

IV.3.3. Le Polymorphisme et le lien dynamique

Comme nous l'avons vu, il existe 2 types de polymorphisme: le premier est l'**Overloading** et le second est le **Polymorphisme Paramétrique** (appelé aussi **Généricité**). Leur principe a été énoncé dans le point concernant la pilosophie orientée objet.

Rappelons que l'**overloading** est la capacité qu'ont deux classes, ou plus, à répondre au même message, chacune à leur façon. Ainsi, dans un langage conventionnel, l'opérateur "+" ou le message "Print" peuvent être utilisés par des occurences de types de données différents. Par exemple, "Print" peut être appliqué pour imprimer, entre autres, un entier ou un "string" de caractères. Dans les langages qui le permettent, l'"overloading" est disponible pour chaque type de données abstrait défini par le concepteur. Il disponible pour n'importe quelle opération de n'importe quel type d'occurence. Ainsi, toutes les classes qui ont une fonction d'impression, exécuteront, à la réception du message "print", une méthode différente afin d'imprimer leur contenu.

Il est nécessaire d'utiliser, en C standard, des fonctions séparées pour effectuer des opérations similaires qui diffèrent seulement par le type des données sur lesquelles elles agissent. C++ permet, quant à lui, d'employer un seul nom de fonction pour déclarer plusieurs fonctions qui se distinguent seulement par le type de données sur lesquelles elles opèrent. Ce mécanisme est appellé **overloading de fonction** en C++. Le compilateur utilise automatiquement la version correcte de la fonction en se basant sur le type des arguments. De la même façon, un opérateur peut être *overloadé* en C++. Un opérateur "overloadé" se comporte différemment avec chaque type de données différent avec lequel il doit travailler.

Le **polymorphisme paramétrique** utilise des classes d'objets comme paramètres dans des déclarations de type générique. Ainsi, une seule

procédure peut être appliquée à des types distincts. Dans certains langages conventionnels, comme en Pascal, un tel polymorphisme existe déjà. Cela signifie qu'un programmeur peut, par exemple, créer, dans une déclaration de type, un type de tableau générique :

Tableau: array [1..10] of element,

où "élément" peut être de n'importe quel type, que ce soit un entier, un tableau ou un record.

En orienté objet, le polymorphisme paramétrique permet la construction de classes abstraites ayant, comme paramètres, des types. Par exemple, "Ensemble [element]" peut représenter une classe qui peut être instanciée à "Ensemble [entier]" ou "Ensemble [personne]". Chaque ensemble partage le code qui est implémenté dans "Ensemble [element]".

Comment un nom de message particulier est-il attaché à une méthode ou une implémentation spécifique ?
Ce phénomène est, concrètement, "déterminé **dynamiquement,** principalement à cause de l'overloading, par l'instance cible du message. "Le lien dynamique signifie que le système lie les sélecteurs des messages aux méthodes qui les implémentent au moment de l'exécution (et non au moment de la compilation." [KHOSH,90] Le lien dynamique est donc, pour le polymorphisme, nettement plus efficace que le lien statique (fait à la compilation).
Toutefois, en C++, le lien des variables ("binding time") peut se faire entièrement, soit à l'exécution, soit lors de la compilation avec les avantages et inconvénients de chaque solution.

IV.3.4. Héritage, spécialisation et sous-typage

Nous décrivons, dans la première partie de ce point, le concept d'héritage propre à la conception orientée objet. Ensuite, nous confrontons les notions de sous-typage et d'héritage, qui sont très souvent utilisées de façon interchangeable.

IV.3.4.1. La notion d'héritage

C'est la partie la plus innovatrice de l'approche orientée objet et n'est en général pas fournie par les langages conventionnels.

L'héritage est l'habileté qu'a une classe d'objet à définir le comportement et la structure de données de ses instances comme un *superset* de la définition d'une ou d'autres classe(s). Il permet donc de concevoir une nouvelle classe d'occurences comme étant le raffinement (ou la spécialisation) d'une autre. Ceci se fait par abstraction des similitudes entre classes et, en spécifiant et en concevant les seules différences pour chaque classe d'objet.

Il existe, dans la conception orientée objet, deux types d'héritage: l'héritage de classe et l'héritage d'instance. En fonction de l'un ou l'autre cas, on hérite respectivement soit des caractéristiques définies dans la classe, soit de celles décrites dans l'occurence.

La classe qui hérite est appelée, selon la terminologie utilisée, la **sous-classe,** alors que la classe de laquelle elle hérite est la **super-classe.** De plus, si une classe possède plus de une super-classe, on parle alors d'**héritage multiple**[3]**.** Dans le cas contraire, on a affaire à l'**héritage simple.** Dans ce dernier cas, les classes sont organisées en hiérarchies (de classes) représentables sous la forme d'un arbre.

En général, une sous-classe hérite des méthodes et des variables à instancier de sa super-classe. Cependant, une méthode ou une structure de données peutt être déclarée, selon les possibilités du langage utilisé, comme étant publique, privée ou visible à la sous-classe. Respectivement, dans le premier cas, n'importe quel client[4] peut y accéder directement, la manipuler ou l'invoquer. Dans le second, aucun client ne peut faire quoi que ce soit. Et dans le troisième cas, la variable à instancier ou la méthode peuvent être directement accédées, manipulées ou invoquées, mais ceci seulement par un client qui est une sous-classe. Dans ce dernier cas, la méthode ou la variable seront considérées comme privées pour les clients qui s'occupent seulement de créer des instances de la classe et de les manipuler via les méthodes.

Finalement, une sous-classe peut hériter de 3 composants de sa super-classe: l'interface[5] , le code d'implémentation des méthodes, et les variables à instancier (la structure de données). Ainsi, si une classe hérite

---

[3] l'héritage multiple n'est pas offert par le C++ emplyé.
[4] de l'instanciation ou de l'héritage.
[5] les noms des méthodes.

de l'interface d'une autre classe, l'interface de la première contient celle de la deuxième.

L'héritage du comportement permet, quant à lui, de partager et de réutiliser le code (de programmation) existant. En ce qui concerne l'héritage de la structure de données, il permet le partage de cette structure parmi les instances de données. La combinaison de ces 3 héritages fournit une modélisation et une stratégie de développement de logiciel très puissantes. En d'autres mots, nous pouvons dire qu'une classe qui hérite d'une autre, est presque identique, du point de vue du comportement et des variables à instancier, à la seconde classe. Cependant, la nouvelle classe inclut quelque chose de plus et est alors dite spécialisée. L'héritage fournit donc un mécanisme de classification avec lequel nous pouvons créer des "taxonomies" de classes.

IV.3.4.2. L'héritage et le sous-typage

"L'héritage est un mécanisme qui permet à des modules d'un logiciel de faire référence à et d'utiliser des modules existants. "Un type T1 est un sous-type d'un type T2 si chaque instance de T1 est aussi une instance de T2." Cependant, selon S.Khoshafian et R.Abnous [KHOSH,90], la relation de sous-typage doit, dans le contexte orienté objet, être mise en relation avec le concept de type de données abstrait.

Nous dirons qu'un type de données abstrait est un sous-type d'un autre si:
- sa structure contenue dans ses variables à instancier est un sous-type de la structure de l'autre type de données;
- son comportement défini par la signature et la spécification de ses méthodes est conforme à celui de l'autre type de données abstrait.

Nous pouvons en conclure, de façon générale, que l'héritage s'occupe de l'implémentation (de la relation de sous-typage) alors que le sous-typage est seulement une relation sémantique entre les classes d'objets. Le lien entre ces 2 notions est donc similaire à celui existant entre la classe et le type de données abstrait.

IV.4.CONCLUSIONS

Dans la théorie orientée objet, les entités principales sont les données, à l'opposé des approches conventionnelles où ce sont les procédures.
Plus précisément, une **classe** est l'implémentation la plus courante du concept de base de **types de données abstraits.** Elle décrit et implémente toutes les méthodes qui caractérisent le comportement de ses instances, et contient les informations qui définissent la structure de ses instances. Cette implémentation et cette structure sont totalement encapsulées (ou cachées) à l'intérieur de la classe. Seuls les noms de ses différentes méthodes sont publiquement accessibles. Ceci permettant à l'implémentation d'être étendue ou modifiée sans affecter, d'une quelconque façon, les utilisateurs de la classe. Une classe peut être vue comme un module. Celui-ci peut être utilisé non seulement dans plusieurs applications, mais également, grâce à l'**héritage**, en vue d'étendre ou spécialiser une autre classe.

Chapitre V

Justification du choix de l'orienté objet et enseignements retirés

V.1. INTRODUCTION

Pour réaliser le développement de "Logiciel BD", nous sommes partis de l'analyse fonctionnelle partielle vus dans les chapitres précédents. Le développement comprend la conception des architectures logiques et physiques, le codage et les tests. Nous pouvions pour cela adopter la manière de procéder se référant à l'approche qui nous a été enseignée en seconde licence[1] [AVLAM, 90]. Nous avons toutefois estimé plus intéressant de mettre en pratique une approche différente, en l'occurence l'approche orientée objet. Nous expliquons alors dans ce chapitre les raisons de notre intérêt pour cette approche appliquée au cas particulier du développement de "Logiciel BD".

---

[1] voir le point V.2.1.

V.2. CONTEXTE

Il faut savoir qu'avant de réaliser la découpe orientée objet de "Logiciel BD", nous ne disposions pas de véritables connaissances de la théorie orientée objet. Nous n'en avions alors que quelques idées fort vagues, bien que certains de ses concepts avaient déjà été énoncés au cours de "Méthodologie et développements de logiciels" de seconde licence [AVLAM, 90]. Nous avouons avoir été quelque peu influencés par la "mode" orientée objet et, malgré notre manque de pratique dans ce domaine, nous avons délibérément choisi de réaliser le logiciel "Logiciel BD" selon cette approche. Ce choix a cependant été motivé par certaines caractéristiques propres à l'approche orientée objet que nous exposons dans les points suivants.

V.2.1. Approche "traditionnelle"

L'approche que nous appelons "traditionnelle" est celle qui nous a été présentée au cours de "Méthodologie et développement de logiciels" [AVLAM,90]. Par "traditionnelle" nous entendons que cette approche était pour nous plus familière que celle orientée objet. Nous sommes cependant convaincu de son originalité par rapport aux approches courantes utilisées à ce jour. Elle consiste en une modularisation par raffinements successifs des traitements. Il faut entendre par là, une modularisation d'un programme conçue, premièrement, à partir de l'analyse fonctionnelle, et deuxièmement, en passant par les étapes de hiérarchisation et de modularisation définies ci-dessous. Cette modularisation permet de réaliser une structuration des traitements en décomposant, par raffinements successifs, un traitement global en traitements de plus en plus élémentaires. Le résultat obtenu finalement est une architecture logique. Celle-ci correspond à la structure du logiciel à développer constituée de composants entre lesquels existent des relations logicielles telles que, par exemples, "utilise", "exporte/importe", "hérite" ou "instancie".

Remarquons toutefois que toute architecture logique n'est pas nécessairement modulaire. Cependant, à cause de diverses contraintes (de qualité du système, organisationnelles,...) à respecter, il est nettement

préférable que l'architecture respecte les principes de hiérarchisation et de modularité.

Structurer de manière hiérarchique revient à organiser le système en niveaux (ou couches) d'abstraction différents et ordonnés selon une relation logicielle particulière. Le résultat est un "squelette" basé sur une seule relation. Formellement, "une structure est hiérarchisée si et seulement si il existe une relation R entre les composants qui permet de définir un certain nombre de niveaux tels que:

1. niveau 0 = {A | il n'existe pas B tq R(A,B)}

2. Niveau i = {A | il existe B E niveau i-1 tq R(A,B)

et si il existe C tq R(A,C),

alors C _ au moins au niveau i-1} " [AVLAM,90].

Modulariser un système revient à identifier, pour chaque niveau, les composants (appelés modules), à spécifier ces modules et à définir, entre modules de même niveau ou de niveaux différents, les occurences des relations autres que celle définie dans la hiérarchie.

"Une structure est modulaire si ses composants et ses relations sont tels que:

1. les attributs de chaque composant peuvent être définis de manière simple et précise.

2. tout composant offre:

- une capacité maximale de cacher l'information

- un faible degré de couplage

- une forte cohésion (intramodule) " [AVLAM,90].

Une différence importante entre une structure hiérarchique et une structure modulaire est que, dans une hiérarchie, chaque arc est une occurence d'une seule et même relation, alors que dans une modularisation, chaque arc est une occurence d'une relation mais pas toujours nécessairement la même.

Une des relations logicielles le plus fréquemment utilisées pour une architecture logique est la relation "utilise" qu'on définit comme suit: "un composant A utilise un composant B si et seulement si la validité de A dépend de la disponibilité d'une version correcte de B".

V.2.2. Motivations du choix d'une conception orientée objet pour "Logiciel BD"

C'est donc en se basant, essentiellement, sur les divers avantages théoriques et pratiques qu'elle offre, que nous avons opté pour une conception orientée objet. Les avantages d'une telle méthode concernent principalement la façon d'atteindre les buts de toute méthodologie de conception et peuvent être compris en lisant, au chapitre IV, la partie concernant la philosophie de l'approche orientée objet.

V.2.2.1. Avantages lors de la phase de conception

Avant de commencer, nous avions pu nous rendre compte dans la littérarture que l'utilisation d'objets rencontre un franc succès dès qu'il est question de graphisme ou d'interface homme/machine. L'explication relève davantage de l'intuition que de la preuvre scientifique: "l'adéquation est parfaite entre un objet graphique et sa "représentation objet", le comportement et les attributs d'un objet à l'écran sont directement traduisibles en "méthodes" (...), les évènements deviennent directement des messages entre objets" [MICSS,90]. Nous avons pu ainsi assez aisément dégager et traiter les différents objets tels que "Dessin", "Mise_En_Page", "Banque d'Images", "Ciseaux", "Colle",... Cette façon de travailler nous est apparue plus naturelle que les autres. En effet, dès les premières discussions avec les animateurs et psychologues, nous avions mis en évidence, explicitement, les divers objets que la personne handicapée mentale pouvait manipuler.

Nous avons aussi appris dans la littérature que, par une bonne mise en oeuvre des différentes caractéristiques orientées objet, le code source produit doit normalement être plus court et plus concis (que celui fourni

par une programmation traditionnelle), la lisibilité du source s'en trouvant alors nettement améliorée. La segmentation du code permet également de réduire la durée de la phase de mise au point et facilite la maintenance étant donné qu'il est possible, en théorie, de tester chaque classe de façon individuelle. Cependant, cette même littérature affirme que pour la réalisation d'un software en orienté objet, la plus grande partie du temps total consacré au projet est utilisée lors de la phase de **conception** *(design)*. Les durées d'**implémentation** et de **test** devant théoriquement être proportionnellement plus courtes que la période consacrée à la phase de conception.

Il s'en suit alors, selon [WIRFS,90], en ce qui concerne la construction d'un logiciel orienté objet, une modification du cycle de vie du logiciel élaboré de façon traditionnelle, et ce au point de vue des phases de conception, d'implémentation et de test:

* *Déroulement de la conception (design).*[2]

Il est nécessaire de disposer, au début de la phase de conception orientée objet, de la spécification des exigences, c'est-à-dire ce que le logiciel peut et ne peut pas faire. A partir de cela, il faut produire une conception dont le résultat final se compose :

- d'un système de classes qui répondent aux exigences;

- d'une description du comportement public des classes;

- des modèles de communication entre ces classes.

Cette conception a pour intérêt d'encapsuler des informations dans des entités qui peuvent être implémentées sans considérer les interactions avec le reste du système. Ce qui permet une répartition efficace de ces entités à implémenter entre différents programmeurs.

* *Déroulement de l'implémentation*

La conception orientée objet facilite, en général, la phase d'implémentation en améliorant, pour le concepteur, la capacité de

---

2 nous verrons, au chapitre VI, une méthode qui prend en charge cette phase de conception.

communication avec le ou les programmeur(s). De plus, l'implémenteur possède au début du codage une meilleure compréhension du problème. Cependant, bien que les langages orientés objet fournissent un meilleur support à l'implémentation de la conception orientée objet, ils ne sont pas strictement nécessaires. Une implémentation écrite dans un quelconque langage de programmation peut aussi bénéficier de ce processus de conception.

* *Déroulement des tests*

Dans une conception orientée objet, les entités du système peuvent être isolées et testées séparément, une à une. Une erreur est ainsi beaucoup plus facilement détectable dans une entité spécifique. De la même façon, la spécification soignée des interfaces entre les entités permet aux testeurs de détecter plus aisément les contradictions entre les "outputs" d'un composant et les "inputs" requis par un autre composant.

V.2.2.1. Avantages après la phase de conception

La découpe orientée objet d'un logiciel respecte les principales qualités[3] qu'un logiciel doit avoir (à savoir les maintenabilité, réutilisabilité et fiabilité). Nous avons conclu qu'en ce sens, elle pouvait être bien adaptée au cas particulier de la réalisation de "Logiciel BD". La conception orientée objet est donc souvent considérée, de nos jours, comme une technique permettant la réalisation de softwares robustes qui peuvent facilement être réutilisés, raffinés, testés, maintenus (entretenus) et étendus. La réutilisation d'un software signifie, selon [WIRFS, 90], que celui-ci sera utilisé comme partie d'un logiciel autre que celui pour lequel il a été initialement conçu. Il est raffiné lorsqu'il est utilisé comme base de définition pour d'autres softwares. Il est testé lorsque son comportement est mis à l'épreuve pour déterminer sa conformité, ou non, aux spécifications du programme. Il est maintenu lorsque des erreurs sont trouvées et corrigées. Enfin, il est étendu lorsque de nouvelles fonctionnalités sont ajoutées à un programme existant.

---

[3] présentées au cours de "Méthodologie de développement de logiciels" [AVLAM,90]

**La maintenabilité et la réutilisabilité*

Bon nombre de défenseurs de la théorie orientée objet prétendent "que cette nouvelle approche de la programmation s'attaque à une des faiblesses congénitales de l'informatique: le manque de productivité dans l'élaboration et la maintenance des logiciels" [INFPC,91]. Ce sont donc 2 des caractéristiques d'un logiciel, à savoir sa maintenabilité et sa réutilisabilité, que l'on propose d'améliorer. Ce qui devrait permettre de ne plus recréer la roue à chaque développement, mais de disposer de bibliothèques de modules toutes prêtes pour la construction de logiciels. C'est également l'avis de Bertrand MEYER [MEYER,87] pour qui, quand on construit un nouveau logiciel, il devrait être possible de commander des composants dans les bibliothèques de modules et de simplement les combiner. Une réutilisabilité maximale est, pour beaucoup d'informaticiens, une caractéristique de logiciels très enviable. La thèse que cet auteur avance est que "la conception orientée objet est la technique, actuellement, la plus prometteuse en vue d'obtenir les buts d'extensibilité et de réutilisabilité" [MEYER,87]. Toutefois, si cette caractéristique de réutilisabilité n'est pas plus répandue, c'est pour des raisons qu'il appelle non-techniques telles que l'adage "non-inventé-ici", le manque de bases de données permettant une recherche efficace du module désiré dans les bibliothèques,... En réalité, pour lui, ces motifs ne constituent que la partie visible du problème. Les principales contraintes sont en fait d'ordre technique. Car il est difficile de concevoir un logiciel réutilisable: un logiciel n'est "jamais vraiment le même, jamais vraiment différent" [MEYER,87] d'un autre. Toutefois, pour Bertrand MEYER, "l'approche orientée objet - définie comme la construction de systèmes logiciels équivalant à des collections structurées d'implémentations de type de données abstraites - fournit un ensemble prometteur de solutions" [MEYER,87]. Et ce, parce que la conception et la programmation orientées objet se basent sur des concepts tels que ceux de générécité, d'overloading, d'héritage (multiple) et de type de données abstraites, qui permettent de construire des éléments de logiciel plus flexibles que ceux fournis par d'autres types d'approche (modulaire, procédurale,...).

V.2.3. Enseignements retirés

Nous pensons avoir conçu une conception logique qui, si nous le voulons, est aisément modifiable. Ceci s'est avéré être le cas lors de l'élaboration de la conception selon la méthode de modularisation suivie [WIRFS,90] (voir Chapitre VI), puisque nous l'avons régulièrement modifiée avant d'arriver à la solution définitive présentée en Annexes.

Nous avons alors construit une conception susceptible d'être aisément maintenue. Elle peut aussi être réutilisée, c'est-à-dire que des classes peuvent être utilisées telles quelles dans un autre logiciel, et étendue en fonction de nouvelles fonctionnalités telles que celles dégagées des améliorations possibles de "Logiciel BD".

Il nous est très difficile de répondre à la question de savoir s'il faut préférer une conception "traditionnelle" à une conception orientée objet. Pour avoir une réponse "tranchée" sur cette question, il aurait fallu réaliser une comparaison théorique et pratique complète des deux approches. Cependant, ce type d'analyse n'entre pas dans le cadre de notre mémoire. Nous pouvons toutefois noter que la méthode orientée objet s'est avérée tout-à-fait adéquate pour notre logiciel graphique et cela, bien qu'elle soit beaucoup plus difficile et plus longue qu'une approche traditionnelle.

En conclusion, l'approche orientée objet touche toutes les étapes de la réalisation d'un logiciel (à savoir la conception, l'implémentation et les tests). Le concepteur d'applications doit modifier son mode de réflexion afin de résoudre les problèmes. C'est ainsi que pour pouvoir espérer bénéficier des divers avantages (principalement méthodologiques), nous avons dû, en tout premier lieu, apprendre à penser "objet" et non plus en termes de modules composés de routines. Nous avons été amenés ainsi à approcher le problème d'une nouvelle manière, ce qui nous a demandé un certain temps d'adaptation, le plus délicat étant de sélectionner les objets réellement intéressants à modéliser, à représenter sous forme de classes d'objet.

V.3. CHOIX DU LANGAGE C POUR L'IMPLEMENTATION

Dans le cas de "Logiciel BD", le langage de programmation utilisé n'est pas un langage dit orienté objet, mais bien le langage C qui est un langage traditionnel.

V.3.1. Justification du choix

A l'origine, nous avions choisi d'implémenter "Logiciel BD" dans le langage C++[4] . En effet, il n'existe pas, pour l'instant, d'autre langage de programmation offrant des facilités orientées objet sur Amiga.

Quant au langage Modula il n'est pas un langage orienté objet, mais bien un langage modulaire et se limite donc à la notion de "module".[MEYER,90].

C'est à la suite de manipulations du langage C++ (cité plus haut) que nous nous sommes rendus compte qu'il comportait certaines anomalies de fonctionnement, notamment dans le préprocesseur qui a pour but de "traduire" le code source C++ en code source C standart. De plus, nous ne disposions pas de la documentation et des informations nécessaires pour utiliser pleinement les capacités graphiques de l'Amiga avec cette version du C++. Après avoir pris contact avec plusieurs personnes[5] ayant déjà utilisé cette version du C++ et ayant, elles aussi, connu certains problèmes , nous avons décidé d'abandonner définitivement le langage C++. Etant tenus par la nécessité de produire un logiciel qui "tourne", et ce dans des limites de temps assez réduites, nous avons décidé d'utiliser le langage C pour implémenter le "Logiciel BD".

Le choix du langage C peut paraître étonnant mais, comme nous l'avons vu plus haut, il n'est pas nécessaire de disposer d'un langage dit "orienté objet" pour bénéficier avantages propres à un processus de conception orienté objet. Ainsi, bien que nous n'ayons pas eu le temps de réellement implémenter la majorité des mécanismes proposés par l'approche orientée objet[6], et ce, pour des raisons de temps, nous pensons

---

4 plus particulièrement la version 1.0 du Lattice C++ sur Amiga.
5 notamment avec Monsieur le Professeur Jim LINDSAY de l'Université de Liège.
6 tels que, par exemple l'héritage, le polymorphisme, ...

que l'implémentation du "Logiciel BD" dans le langage C a respecté dans une grande mesure la conception orientée objet réalisée. Nous allons voir dans le point suivant comment il aurait été possible d'implémenter certains des mécanismes orientés objet dans le langage C.

V.3.2. Le langage C et l'approche orientée objet

Il est possible d'appliquer au langage C une technique d'accès discipliné aux structures de données, en s'assurant que ces structures de données sont manipulées uniquement au travers de fonctions. Il faut savoir, qu'en C, toutes les routines sont des fonctions. En plus de cela, c'est la notion de fichier source C qui permet de mieux implémenter des modules. En effet, un fichier, en C, correspond à une unité de compilation et peut contenir plusieurs fonctions et plusieurs données. Certaines de ces fonctions peuvent être cachées à d'autres fichiers, et d'autres peuvent être rendues publiques. Normalement, dans une collection de fichiers sources C (modules), les fonctions définies sont des objets externes, leur nom est connu de façon globale dans tous les fichiers. Il revient donc au programmeur de veiller à éviter les duplications de noms. De plus, il est fort tentant pour le programmeur de déclarer les variables de façon externe afin de permettre la communication inter-modules. Le problème est que ces variables sont aussi déclarées de façon globale, et il faut prendre soin à ce qu'elles soient déclarées de façon unique.

Il est donc nécessaire de trouver un mécanisme pour cacher les objets (variables ou fonctions), pour déclarer privés ces objets, et ne rendre publiques que les objets qui doivent être accédés par d'autres modules. La technique suivante permet de mettre en oeuvre l'encapsulation et d'appliquer par là un certain niveau de modularité: un fichier peut contenir tous les éléments correspondant à l'implémentation d'un ou plusieurs objets abstraits, ou d'un type de donnée abstrait.

K. Dutta [DUTTA, 85] propose la solution qui suit. Cet auteur propose d'utiliser des fonctions et des variables déclarées *static* en C car elles possèdent la propriété nécessaire; elles sont invisibles aux autres modules (fichiers sources). "En C, *static* n'est pas seulement synonyme de permanence mais est aussi synonyme de ce qu'on pourrait appeler *privacy."* [DUTTA, 85] Les objets statiques externes (variables ou

fonctions) sont seulement connus à l'intérieur du fichier source dans lequel ils apparaissent, et leur nom n'interfère pas avec les variables ou les fonctions de même nom dans d'autres fichiers.

On peut ainsi, pour rendre plus explicite l'approche utilisée, deux synonymes pour les mots-clés "extern" et "static". Ces deux synonymes sont respectivement "PRIVATE" et "PUBLIC". On les définira de la façon suivante en C:

#define PRIVATE static
#define PUBLIC extern

Ces deux synonymes devraient normalement faire partie d'un fichier "header" afin d'être introduits dans chaque fichier source de module, et être manipulés par le préprocesseur du compilateur C. Ces fichiers "headers" décrivent des structures de données partagées. Chaque fichier source nécessitant ces structures de données y auront accès au travers d'une directive "include"[7] et qui est de la forme: *#include <header.h>* où *header.h* est le nom du fichier "header". Cela revient à copier le contenu de tout le fichier "header" à l'endroit où cette directive apparaît, et permet au fichier incluant d'accéder directement à la définition des structures contenues dans *header.h* .

La convention est que maintenant, chaque objet (variable ou fonction) déclaré de façon globale doit être déclaré "PUBLIC" ou "PRIVATE". Les objets "PRIVATE" sont entièrement locaux au module qui les définit, alors que les modules "PUBLIC" peuvent être référencés dans d'autres modules. Bien sûr, tous les objets dans un module, qu'ils soient "PRIVATE" ou "PUBLIC", doivent être déclarés de façon unique.

Cette approche simple du masquage des objets au niveau des modules permet au programmeur d'utiliser arbitrairement les noms d'objets "PRIVATE" et de s'assurer que ce sont seulement les objets qui doivent l'être qui sont rendus "PUBLIC".

Certaines caractéristiques plus spécialisées de C fournissent d'autres possibilités en ce qui concerne une véritable approche orientée objet. Un des aspects fondamentaux de la programmation orientée objet est que l'on

---

7 prise en charge par le préprocesseur C.

peut voir chaque objet regroupant avec lui toutes les opérations qui lui sont applicables, et cela durant toute l'exécution. En C, les instances de types de structures[8] peuvent contenir, parmi leurs champs, des référence (pointeurs) vers des fonctions.

exemple:

```
typedef struct [
        int dernier;
        float contenu[MAXSIZE];
        void (*pop) ();
        void (*push) ();
        float (*sommet) ();
        BOOL (*vide) ();
] PILE
```

Les deux premiers composants de la structure sont un entier et une table. Les autres sont des références (pointeurs) à des fonctions. On utilise le mot-clé *float* en C pour définir un réel. Chaque instance de type doit être initialisée afin que les champs concernant les références pointent vers les fonctions appropriées. B. Meyer [MEYER, 88] propose la solution suivante: si *Une_Pile* est une variable du type *PILE,* et *C_Pop* une fonction retirant un élément de pile, on assignera au composant *pop* de la structure *Une_Pile* la référence à cette fonction de la façon suivante:

```
Une_Pile.pop = C_Pop
```

La fonction C_Pop doit avoir accès à l'objet Pile approprié; d'où il a besoin d'un argument. Cette fonction doit être déclarée de la façon suivante:

```
C_Pop(s)
PILE s;
[
... Implémentation d'un opération Pop ...
]
```

8 équivalentes à des records en Pascal.

Afin que *Pop* puisse être appliqué à un objet de type pile *Une_Pile,* on fera:

Une_Pile.pop(Une_Pile)

Cette technique peut fonctionner dans une certaine mesure. Elle peut même être étendue afin d'imiter l'héritage. Mais elle n'est applicable a aucun développement sérieux: elle implique, en effet, que chaque instance de classe contienne physiquement des références à toutes les routines qui peuvent y être appliquées. Ainsi l'espace d'"overhead" serait prohibitif, spécialement avec l'héritage. Il est possible de diminuer cet espace à un niveau acceptable si on tient compte du fait que les routines sont communes à toutes les instances d'une classe. On pourrait donc introduire pour chaque classe une structure de donnée "run-time", le descripteur de classe, contenant les références aux routines de cette classe. Ce descripteur peut être implémenté comme une liste chaînée ou une table. A partir de ce moment les instances de classe ne nécessitent qu'une seule référence au descripteur de classe, plutôt que de référencer individuellement chaque routine.

Ces exemples montrent qu'il est possible de trouver des techniques d'implémentation pour émuler la programmation orientée objet en C. Mais cela ne veut pas dire pour autant qu'il faille utiliser ces techniques à tout prix. Car le danger d'essayer de forcer l'utilisation de concepts orientés objet en C est d'obtenir une construction inconsistante et de détériorer le processus de développement de logiciels et la qualité des produits résultants. "Une approche hybride à pour résulat une qualité hybride" [MEYER, 88]. Pour bénéficier des techniques orientées objet, on doit utiliser un encadrement compatible, et surmonter les limitations des langages comme C qui - sans tenir compte de ses autres caractéristiques - a été conçus à des fins toutes autres.

V.4. CONCLUSIONS

Pour pouvoir profiter de la pleine puissance de la théorie orientée objet, il faut bien entendu réaliser une conception selon une méthode orientée objet. De plus, l'idéal serait de réaliser l'implémentation du programme dans un langage dit orienté objet, c'est-à-dire qui met en oeuvre au moins les différents concepts orientés objet tels que l'encapsulation, l'overloading, la généricité,.... Mais une bonne implémentation respectant la conception orientée objet dans un langage dit "traditionnel" peut aussi bénéficier d'une réutilisabilité accrue du code (et des modules) et d'une meilleure flexibilité des programmes. A côté de ces avantages non-négligeables, l'approche orientée objet exige, de la part du concepteur, une bonne maîtrise de ses concepts, ce qui renforce la complexité de la tâche de conception et peut, par conséquent, augmenter considérablement le temps de réalisation de cette conception.

Le logiciel "Logiciel BD" n'a, quant à lui, pas été implémenté dans un langage orienté objet et nous n'avons ainsi pas pu implémenter tous les mécanismes permettant de mettre en oeuvre la philosophie orientée objet. Cela ne signifie par pour autant que ce logiciel ne jouisse pas d'un bon degré de maintenabilité et de réutilisabilité.

Chapitre VI

Méthode de conception orientée objet suivie pour la réalisation de "Logiciel BD"

VI.1. INTRODUCTION

Nous décrivons, dans ce chapitre, la méthode orientée objet que nous avons suivie pour réaliser la découpe orientée objet de "Logiciel BD". Nous explicitons ainsi les différentes étapes par lesquelles nous avons dû passer avant d'obtenir le résultat final de cette découpe. Chacune de ces étapes est complétée par des exemples émanant directement de la conception orientée objet finale de "Logiciel BD" [1].

VI.2. LA METHODE SUIVIE

La méthode référencée pour la conception du "Logiciel BD" est celle proposée dans l'ouvrage "Designing Objet-Oriented Software" [WIRFS,90]. Il existe différentes méthodologies de conception de logiciels en orienté objet. Nous ne sommes toutefois pas en mesure

---

[1] La conception complète se trouve en Annexes.

d'affirmer qu'il en existe une reconnue par tous comme étant la meilleure ou comme étant le standart à suivre. Chacune de ces méthodes possède ses propres caractéristiques et a été développée pour rencontrer des besoins spécifiques à des situations particulières. De plus, la plupart d'entre-elles, étant donnée leur spécificité, sont souvent beaucoup trop lourde si on veut les appliquer au cas particulier du "Logiciel BD".

Nous citons, ci-après, quelques méthodologies parmi les plus connues et dont nous avons eu connaissance en consultant la littérature:

"- **ObjectOry** et **Booch** qui sont des méthodologies "lourdes", très formalisées, multi-langages et adressant tous les stades du génie logiciel.

- **OBA** ou Object Behavior Analysis qui est une méthodologie d'analyse souple et particulièrement efficace.

- **OpenTalk** qui est une méthodologie d'approche incrémentale de la conception, de l'implémentation et de la production du logiciel." [MCSYS,90].

Nous avons opté pour la méthode sus-mentionnée [WIRFS,90] car elle était la seule pour laquelle nous disposions de la documentation nécessaire à son application. En outre, elle possède l'avantage de ne pas être aussi lourde que celles citées précédemment. Cette méthode est un moyen d'obtenir une architecture logique et physique prêtes à l'implémentation. Elle n'est pas strictement linéaire car les étapes qu'elle propose peuvent être reprises ultérieurement s'il s'avère nécessaire d'effectuer des modifications au résultat de celles-ci.

VI.2.1. Fonctionnement de la méthode

La méthode se décompose en deux phases consécutives.
La première phase est la phase d'exploration qui se compose, premièrement, de la définition des classes, deuxièmement, de l'identification des responsabilités et, troisièmement, de la description des collaborations.
La phase suivante est la phase d'analyse qui devrait aboutir sur une conception solide et fiable. Pour cela, il faut analyser et améliorer la structure des hiérarchies de classes et ce, afin de maximiser la réutilisation du code et de construire un logiciel facile à maintenir et à étendre. C'est aussi à ce niveau qu'il faut analyser les collaborations entre les classes et identifier les sous-systèmes afin d'améliorer l'encapsulation.

VI.2.2. Les différentes étapes

Nous présentons dans ce point chaque étape proposée par la méthode que nous avons suivie.

VI.2.2.1. Elaboration d'un scénario

Cette première étape consiste à rédiger un scénario complet du futur logiciel. Ceci, afin de mettre en évidence tous les éléments utiles à l'élaboration du schéma de base qui sera composé des différentes classes. Cette étape n'était pas nécessaire en ce qui concerne notre logiciel, car nous disposions déjà de l'énoncé final de "Logiciel BD" ainsi que son analyse fonctionnelle (partielle) présentés aux chapitres II et III.

VI.2.2.2. Identification des classes

La seconde étape consiste à trouver, à partir du scénario, toutes les classes du système appelées, à ce stade, classes candidates. L'ensemble composé de ces classes constituera le pilier de base de toute la conception. C'est en fait autour de cet ensemble que tout le reste viendra se greffer.

Pour dégager une première liste de classes:

* 1. on modélise les objets physiques (ex. imprimante, souris,..) et les entités conceptuelles (ex. fenêtre, outil,..);

* 2. on évite les synonymes;

* 3. si l'emploi d'un adjectif engendre un comportement différent de l'objet, alors on crée une nouvelle classe;

* 4. on modélise les catégories de classes, les interfaces externes (l'interface utilisateur) et les valeurs des attributs d'un objet, mais pas les attributs eux-mêmes.

C'est ainsi qu'il a été aisé de trouver des classes telles que "Dessin", "Case", "Mise_En_Page", "Souris", "Banque_Image",... qui découlaient directement de l'énoncé final du logiciel.

Ensuite, en se basant sur cette liste de classes candidates, la méthode propose d'identifier "le plus possible" de classes abstraites. Rappelons qu'une classe abstraite existe, par définition, pour contenir un comportement commun à plusieurs classes.
Nous avons donc regroupé certaines classes comme "Image", "Bulle", "Commentaire",... dans une classe abstraite commune que nous avons appelée "Elément_Dessin" et qui regroupe le comportement et les données communes à ces classes.

VI.2.2.3. Identification des responsabilités.

La troisième étape nous permet de définir le rôle que chaque classe jouera dans le système: c'est ce qu'on appelle, dans la méthode, les responsabilités de la classe. Une responsabilité comprend la connaissance qu'une instance maintient et les actions qu'elle peut réaliser. Ces responsabilités sont, soit d'ordre publique[2], soit d'ordre privé [3].

---

[2] C'est-à-dire accessibles par d'autres instances.
[3] C'est-à-dire qui ne peuvent être requises par d'autres occurences.

Au départ, pour identifier les différentes responsabilités, il faut se baser sur les spécifications et le scénario du logiciel ainsi que sur les classes déjà définies. A partir de ces sources, on peut découvrir des actions que des objets (du système) doivent exécuter.Il est également possible d'identifier des responsabilités en examinant, principalement, entre les classes, les relations suivantes:

- "**est de type de**" qui fait paraître des relations d'héritage,

- "**est analogue à**" qui permet de trouver des super-classes,

- "**fait partie de**" qui permet d'obtenir des classes.

Toute responsabilité identifiée doit ensuite être logiquement rattachée à une ou plusieurs classes. Il faut, pour ce faire, veiller à:
* 1. répartir équitablement, entre classes, l'intelligence du système. On obtient ainsi un système plus flexible et donc plus facile à modifier.
* 2. rendre les responsabilités aussi générale que possible.
* 3. conserver le comportement lié à des informations. Ainsi, si une classe dispose de certaines informations, il est normal de lui assigner les responsabilités les manipulant.
* 4. garder, en un seul endroit, les informations relatives à une chose. Les responsabilités du maintien de certaines informations ne devraient donc pas être partagées, afin d'éviter une duplication pouvant mener à une incohérence.
* 5. partager des responsabilités. Une responsabilité peut être divisée et partagée entre plusieurs classes.

A ce stade, il faut donc savoir *qui* est responsable de *quoi*. C'est pourquoi, il est intéressant de se poser plusieurs questions telles que: "Que se passe-t-il quand ou si...?", et d'y répondre. Les classes qui ne disposent pas encore, à ce moment, de responsabilités sont alors examinées. Après examen, soit la classe a au moins une responsabilité, soit elle n'en a pas mais elle est quand même conservée, soit elle est détruite car elle n'a plus de raison d'être.

Quelques responsabilités dégagées pour la classe "Dessin":
* Envoyer le contenu graphique du DESSIN à l'IMPRIMANTE.
* Afficher à l'écran le contenu graphique du DESSIN à l'écran.
* Mettre le DESSIN à "l'état actif".

Ou pour la classe "Case":
* Mettre la CASE à "l'état actif" .
* Fournir la couleur de fond de la CASE.
* Fournir les dimensions de la CASE.
* Afficher la CASE à l'écran (en tenant compte de son état).

VI.2.2.4. Identification des collaborations

La quatrième étape doit permettre de savoir comment les classes vont réaliser leurs reponsabilités. En fait, soit la classe réalise elle-même les opérations nécessaires, soit elle collabore avec d'autres classes. On parlera alors de collaborations entre classes.
Nous dirons qu'une classe collabore avec une autre si, pour réaliser une responsabilité, elle a besoin d'envoyer un message à l'autre classe.

Pourquoi les collaborations sont-elles si importantes? Le propos des collaborations, dans l'application, est de révéler le flux du contrôle et des informations durant son exécution; d'où une meilleure prise de décisions quant à la conception de l'application. La mise en évidence d'une telle communication permet d'identifier des sous-systèmes[4] de classes collaborantes. Trouver ces sous-systèmes est important pour, ultérieurement, encapsuler le comportement et la connaissance dans la conception.

Pour identifier des collaborations, on se posera, pour chaque responsabilité, les questions suivantes :
* 1. La classe, est-elle capable de remplir cette responsabilité elle-même?
* 2. Si non, de quoi a-t-elle besoin?
* 3. Ces besoins, de quelles classes peut-elle les obtenir?

---

[4] Cfr. infra

De même, pour chaque classe, on peut se demander :
* 1. Que fait ou connaît cette classe?
* 2. Quelles autres classes ont besoin du résultat ou de l'information de ses responsabilités?
* 3. Existe t-il une classe qui n'interagisse avec aucune autre, c'est-à-dire que la classe ne collabore avec aucune autre classe et qu'une quelconque autre classe ne collabore avec elle? Si oui, cette classe doit être mise à l'écart.

Il est également possible d'examiner des relations entre classes afin d'identifier des collaborations. Les 3 relations les plus courantes, outre les relations spécifiques à l'application, sont "fait partie de", "a la connaissance de" (des classes peuvent connaître certaines choses à propos d'autres classes, même si ces dernières ne font pas partie des premières) , et "dépend de".

Exemples:

Pour la classe "Mise_En_Page", nous avons trouvé que:

* pour réaliser la responsabilité "Afficher la Mise_En_Page à l'écran", la classe "Mise_En_Page" a besoin de collaborer avec la classe "Case" .

* pour réaliser la responsabilité "Fournir la Case active", la classe "Mise_En_Page" a besoin de collaborer avec les classes "Contexte_Réalisation" (qui regroupe toutes les informations concernant l'état du logiciel à un moment donné) et "Case" .

VI.2.2.5. Identification des hiérarchies

Cette cinquième étape met à la disposition du concepteur différents outils (les graphes hiérarchiques et les contrats), à la fois graphiques et conceptuels, qui vont lui permettre d'avoir une vision globale des relations d'héritage du système.

* Les graphes hiérarchiques.

Cet outil permet de représenter graphiquement les relations d'héritage entre les classes. Une classe y est représentée par un rectangle libellé par le nom de la classe. De plus, par convention, le coin supérieur gauche est noirci lorsque nous avons affaire à une classe abstraite. L'héritage, quant à lui, est indiqué par une ligne reliant la super-classe et la sous-classe.

A partir des résultats obtenus grâce à cet outil, il est possible d'analyser la relation d'héritage entre classes afin d'identifier et de résoudre des problèmes dans la conception.

Pour ce faire, il faut :

- modéliser une hiérarchie "est de type de": chaque classe devrait être un type spécifique de sa super-classe et, ainsi, supporter au moins toutes les responsabilités définies par sa super-classe;
- factoriser les responsabilités communes le plus haut possible dans la hiérarchie en créant, éventuellement, une classe abstraite;
- s'assurer que les classes abstraites n'héritent pas de classes concrètes, car une classe abstraite supporte ses responsabilités indépendemment de l'implémentation;
- éliminer les classes qui n'ajoutent pas de fonctionnalité, sauf si la classe implémente de façon unique une responsabilité dont elle hérite.

Nous présentons, comme exemple, ci-après, le graphe hiérarchique "Objet_BD_Affichable" suivi de la description de 3 de ses classes.

Graphe hiérarchique de Objet_BD_Affichable:

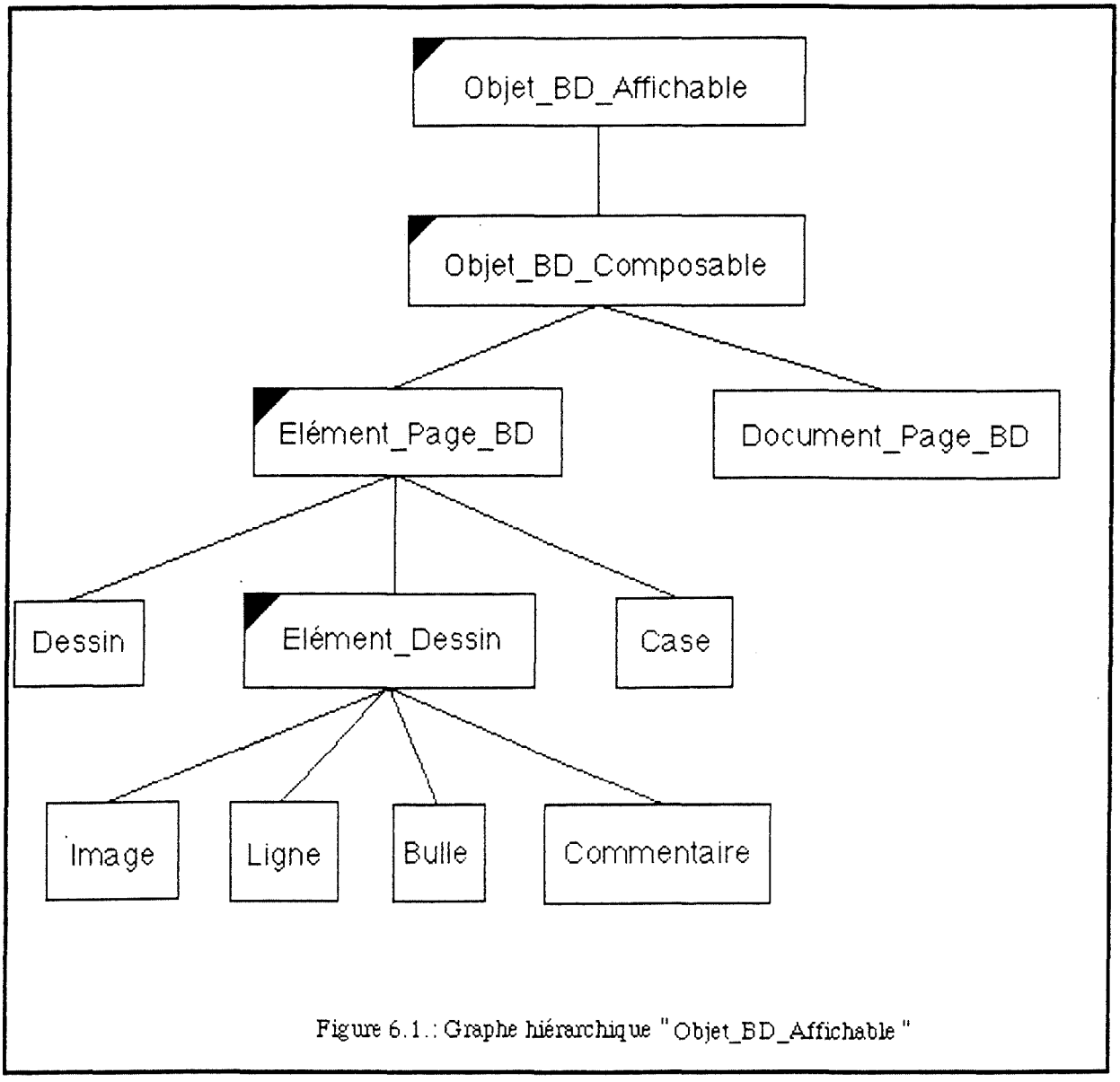


Figure 6.1.: Graphe hiérarchique "Objet_BD_Affichable"

Classe concrète: **Dessin.**
Super-Classe(s): Elément_Page_BD.
Sous-Classe(s): Aucune.
Graphe Hiérarchique: Objet_BD_Affichable.
Description: Un dessin est le contenu graphique d'une case de la page de BD. Il s'agit d'une collection (liste) d'ELEMENTS DESSIN qui ne peut contenir, à la fin des manipulations du DESSIN par l'UTILISATEUR, qu'un seul ELEMENT. L'unique ELEMENT sera une IMAGE de forme rectangulaire dont les dimensions sont celles de la CASE à laquelle appartient le DESSIN.

Classe concrète: **Case.**
Super-Classe(s): Elément_Page_BD.
Sous-Classe(s): Aucune.
Graphe Hiérarchique: Objet_BD_Affichable.
Description: Cette classe représente une CASE d'une MISE EN PAGE. Il s'agit du contour rectangulaire du DESSIN correspondant dans la PAGE de BD.

Classe abstraite: **Elément_Dessin.**
Super-Classe(s): Elément_Page_BD.
Sous-Classe(s): Image, Bulle, Commentaire.
Graphe Hiérarchique: Objet_BD_Affichable.
Description: Cette classe représente un élément constitutif d'un DESSIN. Un élément d'un DESSIN est un élément de la liste gérée par ce DESSIN.

* Identification des contrats

Un contrat définit un ensemble de requêtes qu'un client peut demander à un serveur qui, dans tous les cas, garantit une réponse à ces requêtes. Du point de vue de la classe, un contrat est un ensemble cohérent de responsabilités.
Pour déterminer les responsabilités qui vont appartenir à un contrat, il faut:
* grouper les responsabilités utilisées par les mêmes clients.
* maximiser la cohérence des classes: tout comme un contrat doit être un ensemble cohérent de responsabilités, une classe doit supporter un ensemble cohérent de contrats.

Dans la description de la classe "DESSIN" qui suit, on fournit quelques contrats (affichés en gras) de cette classe ainsi que leurs responsabilités.

Classe concrète: **Dessin**
Contrats:
Manipuler la liste des ELEMENTS du DESSIN.
* Ajouter un nouvel ELEMENT DESSIN dans la liste.
* Détruire l'ELEMENT DESSIN de Numéro d'Ordre donné dans la liste.
* Sauvegarder en mémoire secondaire la liste des ELEMENTS DESSIN.
* Restaurer la liste des ELEMENTS DESSIN.

Gérer l'affichage du DESSIN sur un support d'affichage.
* Envoyer le contenu graphique du DESSIN à l'imprimante.
* Afficher le contenu graphique du DESSIN à l'écran.

Gérer l'état du DESSIN.
* Mettre le DESSIN dans l'état actif.
* Désactiver le DESSIN.

VI.2.2.6. Identification des sous-systèmes

Cette sixième étape permet d'évaluer des sous-systèmes qui représentent des groupes de classes, et auxquels sont attribués des responsabilités.

Un sous-système est un groupe de classes, ou un groupes de classes et d'autres sous-systèmes, qui collaborent afin de supporter des contrats. D'un point de vue externe au sous-système, celui-ci est perçu comme un ensemble très soudé, où chaque élément qui le compose travaille en étroite collaboration afin de fournir une fonctionnalité clairement délimitée. De l'intérieur, il se compose de classes et de sous-systèmes qui, tous deux, collaborent l'un l'autre pour supporter divers contrats contribuant au comportement général du système.
Cependant, un sous-système n'est qu'une unité conceptuelle: il n'existe pas durant l'exécution. Ce n'est donc pas le sous-système qui "remplit" un contrat, mais c'est une classe qui réellement le supporte.

Nous avons pu ainsi dégager cinq sous-systèmes pour le logiciel et que nous avons nommés et définis comme suit:

- DOCUMENT PAGE BD: ce sous-système encapsule les classes responsables des détails de représentation d'une page de bande dessinée.

- EDITEUR GRAPHIQUE: ce sous-système encapsule les classes représentant les divers outils graphiques offerts à l'utilisateur pour ce qui concerne le travail dans le dessin d'une case de la page de bandes dessinées.

- INTERFACE UTILISATEUR: ce sous-système encapsule les classes permettant de contrôler le dialogue entre l'utilisateur et l'ordinateur.

- GESTION B.I.: ce sous-système encapsule les classes responsables des détails relatifs à la représentation et à la manipulation des banques d'images.

- GESTION UTILISATEUR: ce sous-système encapsule les classes responsables des informations et des manipulations concernant les utilisateurs de "Logiciel BD".

Chacun des 5 sous-systèmes définis ci-dessus est composé d'une série de graphes hiérarchiques. Toutefois, nous nous limitons, à titre exemplatif, au seul sous-système "Document Page BD" en ne fournissant que quelques uns de ses contrats.

Sous-système **DOCUMENT PAGE BD**:

Contrats:

1. S'afficher.
Serveur: Classe Objet_BD_Affichable.

2. Mettre à jour les documents page BD résidant dans le système (Zone Travail d'un UTILISATEUR).
Serveur: Classe Gestionnaire_Document_Page_BD.

3. Mettre à jour les mises en page BD résidant dans le système (Zone Travail d'un UTILISATEUR).
Serveur: Classe Gestionnaire_Mise_En_Page.

VI.2.2.7. Identification des protocoles et signatures

Grâce à cette septième étape, il est possible de s'assurer que les responsabilités sont suffisamment raffinées et que les messages sont bien nommés. Elle a pour but d'essayer de maximiser la réutilisabilité du design et de minimiser sa taille conceptuelle, en considérant de façon critique l'emploi de l'héritage.

Pour cela, il faut:
-1. construire des protocoles pour chaque classe, c'est-à-dire les signatures spécifiques des méthodes que chaque classe implémente.

-2. écrire une spécification complète de conception pour chaque classe, sous-système et contrat.

Une fois les responsabilités assignées aux classes, et les classes et sous-systèmes définis, on spécifie de façon très précise les classes. Jusqu'ici, la conception des classes a été modélisée en termes de contrats, responsabilités privées, clients et serveurs. Maintenant, la méthode propose de passer des contrats aux protocoles où chaque protocole est un ensemble de signatures auxquelles une classe doit répondre. Pour ce faire, premièrement, il est nécessaire de généraliser les protocoles et, deuxièmement, de définir raisonnablement les valeurs par défaut.

En résumé, dans cette étape, il faut traiter les responsabilités de chaque classe ou sous-système de l'application, en leur associant en un ensemble de signatures. A chaque responsabilité est (sont) ainsi associé(s) un ou plusieurs message(s). Ces derniers doivent être nommés "intelligemment", et une spécification des types des arguments requis et du type d'instance retournée par la méthode (si nécessaire) y est jointe.

Exemple:

Classe concrète: **Document_Page_BD.**
Super-Classe(s): Objet_BD_Composable.
Sous-Classe(s): Aucune.
Graphe Hiérarchique: Objet_BD_Affichable.
Description: Cette classe représente une PAGE DE BD. Elle représente la structure regroupant différents ELEMENTS de PAGE BD. Un DOCUMENT PAGE BD contient une liste de DESSINS. Chaque dessin qui la compose a une position unique dans cette PAGE BD. Cette position est fournie par la MISE EN PAGE associée au DOCUMENT PAGE BD. Le nombre de DESSINS et leur position dans la page blanche reste fixe (ils sont imposés par la MISE EN PAGE associée).

Contrats:
Mettre à jour et connaître les informations concernant les DESSINS.
Connaître et manipuler le DESSIN actuel.
Get_Dessin_Actuel() renvoie un DESSIN.
Utilise: Contexte_Réalisation, Mise_En_Page, Dessin.
Cette méthode fournit la référence du DESSIN actuel.
Set_Dessin_Actuel(Dessin).
Utilise: Contexte_Réalisation, Mise_En_Page, Dessin, Case.
Cette méthode permet de sélectionner un nouveau DESSIN actuel.

Connaître et manipuler les informations concernant tous les DESSINS.
Get_Dessin(Numéro_Case) renvoie un DESSIN.
Utilise: Mise_En_Page, Dessin.
Cette méthode fournit la référence du DESSIN se trouvant dans la CASE de numéro donné de la MISE EN PAGE associée.
Get_Dessins_Non_Vide() renvoie un Entier.
Utilise: Mise_En_Page.
Cette méthode fournit le nombre de DESSINS non vides contenus dans le DOCUMENT PAGE BD.

(...)

VI.3. CONCLUSIONS

Nous venons de présenter, en l'illustrant par des exemples provenant directement de notre conception, la méthode orientée objet proposée dans l'ouvrage "Designing Object-Oriented software" dont nous nous sommes inspirés lors de la réalisation de la conception orientée objet de "Logiciel BD". Durant l'élaboration de cette conception, il a souvent été nécessaire de revenir sur certaines des étapes proposées par la méthode afin d'affiner à chaque fois la solution. De plus, pour chaque modification faite dans une étape, nous avons dû répercuter tous les changements dans les étapes suivantes. Une telle façon de travailler a donc nécessité beaucoup de temps et peut à la longue être fort fastidieuse.

Nous avons cependant préféré conserver une certaine autonomie par rapport à la méthodologie en n'appliquant pas de façon rigoureuse tous les points de chaque étape. En effet, la méthode n'offre pas un "algorithme" permettant d'arriver à l'unique solution optimale et la suivre aveuglément, pas à pas, peut aboutir à une solution trop théorique où tout le système est modélisé sous forme de classes. Une telle solution n'est pas envisageable si l'on veut réaliser une implémentation sur ordinateur. Ainsi avons-nous accordé plus d'importance à certaines étapes qu'à d'autres et appliqué les critères de façon sélective en tenant compte des nécessités premières dues à l'implémentation.

La méthode offre des outils de modélisation fort utiles pour représenter de façon claire le contenu de la conception. Ainsi, les graphes hiérachiques et la découpe en sous-systèmes nous ont-ils aidés à mieux "visualiser" les divers éléments qui constituent notre découpe. Les nombreux critères et règles proposés à chaque étape aident aussi à améliorer la qualité de la conception, mais les appliquer tous en même temps peut être très difficile. Nous avons donc été contraints d'opérer des choix quant à leur application.

En ce qui concerne le contenu même de la découpe orientée objet de "Logiciel BD", nous avons veillé à ce que chaque classe et chaque responsabilité soient les plus générales possible, afin que de futures modifications des spécifications puissent se faire sans devoir la

recommencer toute. Ainsi par exemple, la classe "Dessin" a été conçue en vue de pouvoir gérer différents "niveaux" d'images à l'intérieur d'un même dessin, alors que l'implémentation ultérieure de cette classe ne permet pas la gestion de ces niveaux.

Chapitre VII

INTERFACE HOMME/MACHINE ET "LOGICIEL BD"

VII.1. INTRODUCTION

La convivialité (ou "User friendliness") d'une interface, moyen par lequel l'utilisateur peut communiquer ou dialoguer avec l'ordinateur, est une notion qui est aujourd'hui considérée avec beaucoup plus d'intérêt qu'auparavant dans le monde du développement des logiciels. Le logiciel de création de Bandes Dessinées, "Logiciel BD" étant destiné à des personnes mentalement déficientes, il est fondamental que son interface soit adaptée à la population qui l'utilisera.

Nous présentons, dans la première partie de ce chapitre, quelques aspects théoriques de la conception d'une interface en décrivant les concepts théoriques et empiriques généraux. Ceci recouvre aussi bien les concepts de tâche et d'utilisateur que les critères de qualité et les règles empiriques d'une interface.
Dans la seconde partie, nous décrivons une série de notions relatives à la réalisation d'une interface pour personnes handicapées mentales. Afin de permettre aux personnes ayant un handicap mental léger ou modéré de manipuler aisément le logiciel, nous avons réalisé une interface adaptée à

leurs besoins. Pour ce faire, nous avons dû faire certains choix quant au contenu des divers écrans ainsi qu'en ce qui concerne l'agencement et la présentation des différents éléments qui les composent. Ces décisions se basent, bien sûr, sur les informations contenues dans les chapitres I et III et ont été prises en respectant les conseils des psychologues et des animateurs avec lesquels nous avons travaillé.
Nous présentons finalement, dans une troisième partie, l'interface qui a été réalisée pour l'application du logiciel qui a été implémentée. Nous y donnons une justification des différentes décisions prises par rapport à la théorie présentée avant.

VII.2. ASPECTS THEORIQUES

VII.2.1. Les concepts théoriques et empiriques généraux de l'interface homme/machine

VII.2.1.1. Description générale du problème

L'ordinateur, et plus particulièrement le logiciel, serait un prolongement du cerveau humain, au sens où il fait appel à des processus cognitifs de l'individu tels que la mémoire, la prise de décision, la perception,...
Le but de l'interface serait alors idéalement, "de permettre à la personne de réaliser le mieux possible la tâche qu'elle a à accomplir, c'est-à-dire avec précision, rapidité et sans efforts inutiles (des efforts étrangers à la nature de la tâche)" [BODAR,90]. De cet idéal, cet auteur met en évidence deux notions extrêmement importantes quant à la réalisation d'une interface, à savoir les concepts d'**utilisateur** et de **tâche.**

En ce qui concerne le concept d'**utilisateur,** nous avons, d'un côté, l'utilisateur qui exprime, en des termes liés à la représentation mentale qu'il se fait du problème, ce qu'il espère pouvoir réaliser. D'un autre côté, c'est l'ordinateur qui résoud ce problème en des termes physiques. L'utilisateur final devra donc faire un effort de traduction de ses termes psychologiques en termes physiques, et vice versa, afin de pouvoir évaluer la réalisation de ses objectifs. "Une interface sera d'autant meilleure qu'elle

requiert peu d'efforts cognitifs (mentaux) de la part de l'utilisateur aux plans de l'exécution et de l'évaluation:

- au plan de l'exécution, une bonne interface devrait fournir des commandes et des mécanismes aussi proches que possible des représentations mentales des utilisateurs.
- au plan de l'évaluation, l'interface devrait fournir des dispositifs de sortie (d'affichage) dont le modèle conceptuel est facilement perçu, interprété et évalué par l'utilisateur" [BODAR,90].

C'est pourquoi, pour réaliser une "bonne" interface homme/machine, il est indispensable de prendre en compte les processus cognitifs des personnes de la population pour laquelle le logiciel est conçu. De cet intérêt dépendra, entre autres, la séquence des actions du logiciel, le style de feed-back à mettre en oeuvre et la manière d'afficher les informations. C'est ce qu'on appelle communément l'**ergonomie du logiciel.** Le but est donc de satisfaire les attentes des futurs utilisateurs. Il est donc nécessaire de connaître les caractéristiques cognitives de l'utilisateur final. C'est à ce titre que la psychologie cognitive (cfr.infra) va nous aider en dégageant ces caractéristiques.

"La psychologie cognitive désigne la partie de la psychologie qui s'intéresse à la manière dont l'homme traite l'information (mémorisation, apprentissage,...)" [BARTH,88]. De par l'étude des concepts qu'elle met en évidence, cette science a un double intérêt. Le premier est, dans la perspective de la psychologie du travail, de permettre de sélectionner les individus adéquats. Le deuxième, exploité par l'ergonomie cognitive, permet "d'adapter les dispositifs matériels et logiciels du système homme/machine aux caractéristiques cognitives des individus" [BODAR,90].

En ce qui concerne le concept de **tâche,** il est essentiel, afin de concevoir convenablement un logiciel, de connaître avec la plus grande précision possible son contenu. En d'autres termes, nous devons savoir exactement ce que les utilisateurs voudraient pouvoir faire. Cela revient alors à décrire et à spécifier, de façon détaillée, les fonctionnalités du système.

Cependant, si l'important est une analyse précise de la tâche, les fonctionnalités et le dialogue de l'interface, qui sont respectivement le "quoi" et le "comment" de l'échange d'information entre l'ordinateur et l'utilisateur, doivent normalement être indépendants l'un par rapport à l'autre. Ceci, afin qu'une même fonctionnalité puisse être assurée par divers dialogues.

VII.2.1.2. L'ergonomie du logiciel

L'ergonomie du logiciel, appelée également **ergonomie cognitive,** est devenue, depuis près de dix ans, de plus en plus une étape indispensable dans la réalisation des logiciels interactifs. Des "défauts" tels que celui du manque d'homogénéité dans la conception de l'interface (dénomination différente, structure incohérente,...), ont pu être comblés grâce à elle. "Elle vise à définir une méthode et des recommandations pour la réalisation d'interfaces conviviales et efficaces" [PETOU,90]. Pour ce faire, toute interface doit être conçue selon les capacités physiques et mentales de l'utilisateur auquel elle s'adresse et en fonction de la tâche que l'utilisateur voudrait pouvoir réaliser.

Pour évaluer la qualité d'une interface, il existe une série de critères. Mais, si des limites peuvent être fournies quant au modèle de l'utilisateur, "on ne peut fournir que des règles empiriques à respecter pour la conception d'une interface" [BODAR,90]. Ces règles vont cependant nous permettre de vérifier les critères en question.
Ces critères et ces règles sont présentés, ci-après, de façon distincte. Ceci, à la fois dans un souci de clarté et de non ambiguïté, mais aussi et surtout, afin de différencier le "ce à quoi on doit arriver" et le "comment l'obtenir".

VII.2.1.2.1. Les critères de qualité

"De façon générale, on peut dire qu'il est essentiel de respecter les objectifs, connaissances, représentations et méthodes des utilisateurs" [SCAPI,86]. Ces paramètres, une fois pris en compte et évalués, vont permettre au concepteur du logiciel de réaliser un programme adéquat et efficace.

Ben Schneiderman [SCHNE,87] a proposé 5 critères qui permettent d'évaluer la qualité, appelée convivialité, d'une interface :

* le **temps d'apprentissage,** par un utilisateur appartenant à la population pour laquelle le logiciel a été conçu, des commandes nécessaires pour réaliser une tâche. Pour des personnes mentalement handicapées, ce temps peut être relativement long.

* la **prise en compte des erreurs effectuées** par l'utilisateur, à savoir la fréquence des erreurs et le temps global de leur correction.

* le **temps nécessaire à l'exécution d'une tâche** par un utilisateur, qui dépend bien sûr de la tâche elle-même.

* la **période de rémanence** durant laquelle un utilisateur conserve la connaissance acquise lors de l'utilisation du logiciel, et qui dépend de l'effort cognitif et du temps d'apprentissage.

* la **satisfaction subjective** à utiliser le système, qui est peut-être le critère le plus difficile à évaluer de par sa caractéristique même de subjectivité. Cette satisfaction devrait être presque immédiate pour une population d'individus ayant une déficience mentale.

La prise en compte de ces facteurs est importante pour à la qualité future de l'interface. Cependant, il est difficile de favoriser, au maximum, en même temps, tous ces critères. Ainsi, si nous voulons, par exemple, réduire de façon drastique le taux d'erreurs effectuées par l'utilisateur, nous allons immanquablement engendrer un accroissement du temps d'exécution.
Il s'agit donc, en fonction de la population visée, de trouver le meilleur compromis entre tous ces critères.

VII.2.1.2.2. Les règles empiriques

Les règles empiriques, communément admises comme étant les **règles d'or,** sont issues d'études empiriques [1] et servent à préciser la notion de convivialité pour une interface homme/machine.

---

[1] C'est-à-dire basées sur l'expérience.

"Il est important de noter que, à posteriori, ces règles empiriques peuvent trouver une justification dans les propositions théoriques de l'analyse de la tâche et du modèle de l'opérateur" [BODAR,90].

Ces règles sont les suivantes :

- 1. la **compatibilité** - au sens très général - dont, entre autres, une terminologie compatible au vocabulaire de l'utilisateur final.

- 2. l'**homogénéité** qui permet d'obtenir un environnement constant ou, autrement dit, que l'interface soit **cohérente** du point de vue de ses concepts, de ses représentations et de leur localisation à l'écran ainsi que de sa terminologie.

- 3. la **concision** qui est, par définition, la production de moyens de communication brefs et expressifs.

- 4. la **flexibilité** qui désigne la faculté d'ajustement de l'interface aux variations de l'environnement. Cette règle est importante car les caractéristiques cognitives des individus d'une population ne sont pas identiquement les mêmes, mais aussi parce qu'elle permet l'ajout de nouveaux "inputs devices".

- 5. les **feed-back** [2] et **guidage** qui permettent d'informer l'utilisateur sur l'état du système. "L'utilisateur doit toujours savoir où il se trouve dans une séance de dialogue, ce qui a été fait, et avoir toujours un moyen de poursuivre le dialogue" [SCAPI,86].

- 6. la **charge informationnelle** de l'utilisateur qui est une règle où la charge de la mémoire humaine à court terme doit être minimale. Cette mémoire étant limitée, il s'agit donc de réduire au minimum la charge mnésique de l'utilisateur, afin de ne pas outrepasser ses limites de mémorisation à court terme, mais aussi pour lui demander le moins d'effort possible.

- 7. le **contrôle explicite** de l'interface. L'utilisateur doit avoir l'impression de contrôler l'interface qui, de plus, doit exécuter uniquement les opérations relatives à ce que demande explicitement l'utilisateur.

---

[2] appelé aussi "retour d'informations".

- 8. la **prévention et la gestion des erreurs** qui doit offrir à l'utilisateur la possibilité d'éviter de faire des erreurs ou de les corriger.

- 9. le **séquencement des actions** qui a trait à la manière dont les actions sont enchainées ou mises en séquence. Il est nettement préférable que toute action ait un début et une fin ininterrompus. Ainsi, à la fin de la séquence, l'utilisateur sait qu'il a fini d'exécuter la tâche et qu'il peut désormais en entamer une nouvelle.

VII.2.2. Une interface pour personnes mentalement déficientes

VII.2.2.1. Introduction

Le "Logiciel BD" est destiné à des personnes souffrant d'une déficience mentale légère ou modérée. Or, nous venons de voir que tout utilisateur doit faire un effort de traduction de termes physiques en termes psychologiques, et vice versa. De plus, l'utilisateur susceptible d'utiliser le "Logiciel BD" est dépourvu de la capacité de symbolisation nécessaire à l'accomplissement efficace et immédiat de cet effort. C'est pourquoi, il est d'autant plus indispensable que l'interface de notre programme soit adaptée aux utilisateurs ciblés, en mettant en oeuvre les critères de qualité et les règles empiriques. C'est la raison pour laquelle nous décrivons ci-après la manière dont certains concepts concernant l'interface homme/machine (les styles de dialogue, l'organisation des écrans, les icônes, le prototypage et l'évaluation du cahier des charges) sont à introduire dans la conception de notre interface. En d'autres mots, nous présentons la façon dont ces notions doivent être implémentées quand les utilisateurs sont des personnes souffrant d'un handicap mental.

VII.2.2.2. Le dialogue

En ce qui concerne les dialogues et les fonctionnalités d'une interface, il a déjà été dit, plus haut, qu'ils devaient être **indépendants** l'un vis-à-vis de l'autre. Et ce, principalement dans le but que toute les fonctionnalités puissent être exécutée au travers de dialogues divers. Cette

nécessité de concevoir de façon indépendante le dialogue et les fonctionnalités est d'autant plus importante que la population des individus handicapés mentaux est fort hétérogène.

Alors que pour disposer de fonctionnalités efficaces,il suffit que le concepteur possède une bonne capacité intrinsèque de conception, il en va autrement pour le dialogue. Pour implémenter ce dernier, à travers un ou plusieurs **styles de dialogue,** il est impératif de tenir compte du ou des modèles de l'utilisateur. C'est à cette seule condition que le dialogue, et par là-même l'interface, sera réalisé efficacement. En conclusion, si, pour la conception de notre interface, il faut tenir compte du fait que les utilisateurs potentiels sont des personnes ayant un handicap mental, il faut aussi prendre en compte la gravité du handicap dont souffre l'utilisateur.

Les différents styles de dialogue qui, dans le contexte du handicap, sont à notre disposition, sont les suivants :

- **"Question/réponse":** ce style de dialogue implique de nombreuses interactions entre l'ordinateur et l'utilisateur. Il laisse peu d'initiative à l'utilisateur, est très peu adapté aux situations complexes, et ne présente pas la démarche globalement mais séquentiellement.

- **"Formulaire":** ce style de dialogue est intéressant lorsque l'introduction de données est nécessaire. Cette saisie des données se fait via des champs libellés. Ce style a comme avantages d'être simple et de fournir toute une série d'indications, ce qui ne nécessite qu'un court apprentissage. Cependant, l'initiative de l'utilisateur est faible.

- **"Langage de commande":** ce style de dialogue convient très bien quand on désire laisser à l'utilisateur beaucoup d'initiative et la possibilité de créer des macro-commandes. Le langage de commande offre en plus la possibilité de guider le dialogue et s'avère être rapide pour des utilisateurs expérimentés. Mais ce style engendre un risque élevé d'erreur et nécessite une formation assez importante.

- **"Langue naturelle":** ce style de dialogue permet à l'utilisateur d'utiliser un langage naturel comme, par exemple, le français. Mais s'il ne requiert qu'une formation minime et peut être rassurant pour l'utilisateur, il crée par contre une lourde charge pour l'ordinateur. De plus, vu l'ambiguïté des langues naturelles, une procédure de clarification doit être mise en place.

Nous n'avons donc jamais réellement affaire à une langue naturelle, mais plutôt à un sous-ensemble de celle-ci.

- **"Manipulation directe":** ce style de dialogue permet à l'utilisateur de manipuler directement un objet en sélectionnant, à l'écran, sa représentation graphique [3]. Ce procédé requiert un apprentissage léger, et le résultat de cet action est immédiatement visible ce qui assure une rétroaction. Ceci diminue les erreurs et laisse une certaine initiative à l'utilisateur. Toutefois, la manipulation directe est plus difficile à programmer qu'un autre style de dialogue.

- **"Menus":** ce style de dialogue contient un ensemble d'options parmi lesquelles l'utilisateur effectuera un choix, une fois le menu affiché. Le menu a pour avantages de se baser sur un principe facilement compréhensible, de bien situer les différents choix, de demander un faible niveau d'apprentissage et pratiquement aucun effort de mémorisation. Mais, pour réaliser une tâche, il est bien souvent nécessaire de passer par plusieurs menus distincts. Ceux-ci seront organisés entre-eux soit en une séquence, soit en une arborescence, soit en une multi-arborescence, soit en un réseau de menus. En ce qui concerne leur présentation, soit un seul est présenté à la fois, soit ils sont affichés ensemble en pavage (séparés l'un de l'autre), soit ils sont superposés partiellement, soit ils sont déroulants, soit ils sont en cylindre.

- **"Panaché":** c'est un ensemble des divers styles précédents requis en fonction du contexte, "l'utilisateur doit toujours savoir où il se trouve dans une séance de dialogue, ce qui a été fait, et avoir toujours un moyen de poursuivre le dialogue" [FRAIT,90]. Mais, afin de ne pas surcharger l'utilisateur, le nombre de styles différents doit être limité. De plus, "les styles choisis doivent être clairement distinguables: une trop grande proximité nuit à la cohérence" [FRAIT,90].

Si l'élément de base du dialogue est le style de dialogue, il reste encore à le structurer. Cette opération consiste à définir les différents niveaux de dialogue et le type d'enchaînement approprié.

---

[3] appelée **icône.**

VII.2.2.3. L'organisation des écrans et les icônes

L'organisation des écrans et le choix des icônes doivent, lors de la conception de l'interface, faire l'objet d'une attention toute particulière. Ceux-ci sont normalement mis en oeuvre par l'implémentation, en fonction du (des) modèle(s) de l'utilisateur, de règles empiriques en vue d'obtenir une interface de qualité. Autant les écrans et les icônes seront attrayants et intéressants, autant les utilisateurs ciblés seront attirés par le logiciel.

Afin d'organiser convenablement le contenu de chaque écran, une série de règles d'ergonomie sont mises à la disposition du concepteur. Tout d'abord, des regroupements logiques peuvent être effectués; regroupements qui peuvent subir un renforcement visuel. Ensuite, l'esthétisme d'un écran et l'homogénéité de la présentation d'un écran à un autre doivent également être pris en compte.

Si maintenant, nous désirons attirer l'attention de l'utilisateur sur un élément précis, plusieurs techniques existent, comme par exemples, l'utilisation d'une certaine couleur, l'emploi de divers degrés d'intensité, du son, d'un feed-back, et la taille des caractères. Mais l'utilisation de ces procédés doit se faire avec précaution, un emploi abusif risquant d'engendrer une certaine perturbation chez l'utilisateur.

Une icône est, par définition, la représentation graphique d'un objet. Son rôle est donc de communiquer de l'information sous une forme graphique. Nous en déduisons que l'utilisation d'icônes, dans un programme, est d'autant plus opportune que l'utilisateur a des difficultés pour lire.
Toutefois, avant d'employer une icône quelconque, il faut s'assurer qu'elle sera comprise par la population cible. De plus, chaque icône doit être rapidement reconnue, mémorisable et discriminable. A cette fin, il est nettement préférable qu'elle constitue, pour les utilisateurs ciblés, une représentation réaliste de leur objet.

VII.2.2.4. L'évaluation du cahier des charges et le prototypage

Le cahier des charges pour un logiciel destiné à des personnes handicapées doit contenir, selon [FRAIT, 90], une spécification très rigoureuse des fonctionnalités du logiciel accompagnée d'une description du dialogue. Ce contenu se base, entre autres, sur le modèle empirique de l'utilisateur final.
Cependant, un cahier des charges n'est que rarement, voire jamais, rédigé de façon complète lors de sa première élaboration. Nous découvrons très souvent, par après, soit lors du prototypage (si il a lieu), soit lors des tests sur le terrain, soit lors de l'utilisation quotidienne, des éléments nouveaux qui peuvent venir le modifier ou le compléter.

Le prototypage est un moyen de vérifier le caractère effectif des spécifications fonctionnelles, c'est-à-dire leur conformité aux besoins de l'organisation. En d'autres mots, nous voulons savoir si elles vont produire les résultats espérés par les analystes, les concepteurs et les utilisateurs. "L'objectif principal du prototypage est de permettre la vérification expérimentale de ce caractère effectif par l'exécution des spécifications, sans devoir attendre leur implémentation finale" [BODPI,89]. En se basant sur le prototypage, nous essayons donc d'évaluer le comportement spécifié afin de déterminer l'équilibre entre ce comportement et celui attendu.

"L'objectif de validation et d'amélioration des spécifications peut être complété par un objectif d'apprentissage et de formation : l'outil de prototypage peut être employé par les utilisateurs finaux, lorsque les spécifications sont stabilisées, comme un moyen pour prendre connaissance de celles-ci et se former au système futur en cours d'implémentation" [BODPI,89]. Cet objectif d'apprentissage peut non seulement porter sur le prototypage des règles de traitement, mais aussi sur celui des dialogues homme/machine.

Le cahier des charges, composé des fonctionnalités et du dialogue, doit être évalué avant de commencer l'implémentation. Cela nécessite, entre autres, un prototype ou, tout au moins, une définition détaillée des écrans et des scénarii. Le prototype est alors vu comme un parcours structuré d'écrans. Quant aux scénarii, ils permettent d'examiner ce que l'utilisateur fait, voit et doit savoir. Cette évaluation se fait par rapport aux objectifs, mais également par rapport aux règles ergonomiques.

VII.3. ASPECTS PRATIQUES

Dans cette partie, nous étudions, sur base de la théorie présentée au point VII.2., l'interface homme/machine de l'application de "Logiciel BD" qui a été implémentée.

VII.3.1. Respect des règles empiriques

Le respect des 9 règles empiriques peut varier d'un logiciel à l'autre. Elles ont pour but principal de fournir une interface adaptée aux besoins de l'utilisateur final. Dans le cadre de "Logiciel BD", nous ne possédons qu'un modèle empirique assez rudimentaire de cet utilisateur. Nous présentons ici, règle par règle, l'importance relative que nous leur avons accordée et la manière dont elles ont été respectées.

VII.3.1.1. La compatibilité

Pour que notre interface soit compatible par rapport à la terminologie quotidiennement utilisée par l'utilisateur handicapé mental, nous avons porté une attention particulière à l'utilisation des termes qui étaient utilisés dans le logiciel. Nous avons veillé à ne pas utiliser de terminologie trop technique ou trop abstraite. C'est ainsi que le terme "Mise en page" a été remplacé par le terme "Disposition des cases", ceci afin d'éviter toute confusion avec le terme "Page de BD".

VII.3.1.2. L'homogénéité, la cohérence

Nous avons essayé d'offrir l'interface la plus uniforme et homogène possible. C'est pour cela, que nous avons décidé de réaliser une interface n'utilisant qu'une seule représentation par concept. Ainsi, par exemple, la possibilité de retour en arrière (c'est-à-dire revenir à l'écran précédent) est toujours représentée par une porte ouverte vers l'extérieur. De plus, chacune des représentations est localisée à un même endroit ou, tout au moins, dans une même zone de l'écran. Par exemple, les boutons seront le plus souvent placé dans la partie gauche de l'écran, et ce pour privilégier

une organisation "gauche-droite" pour tous les écrans de l'application. Mais, si un concept doit normalement avoir une seule représentation, il en va de même pour sa terminologie. C'est la raison pour laquelle nous n'utilisons aucun synonyme, ni aucune abréviation pour décrire ces concepts.

Tandis que certaines actions, comme choisir entre la création d'une ancienne ou d'une nouvelle page de bande dessinée, se font à l'aide d'une commande d'un menu, d'autres actions sont réalisées par une manipulation directe, comme par exemple le "crayonnage" à main levée dans une image. De plus, chaque commande représentée par un même texte ou un même graphique produit un résultat équivalent. En ce qui concerne la localisation des concepts, il n'existe que 3 différents types d'écran.

Le premier type d'écran (Figure 7.1.) comprend 3 zones qui servent, respectivement, par ordre croissant, à contenir la "case-mémo" (1), à afficher les boutons (icônes) représentant les outils de manipulation graphique disponibles (2), et à créer un dessin (3).

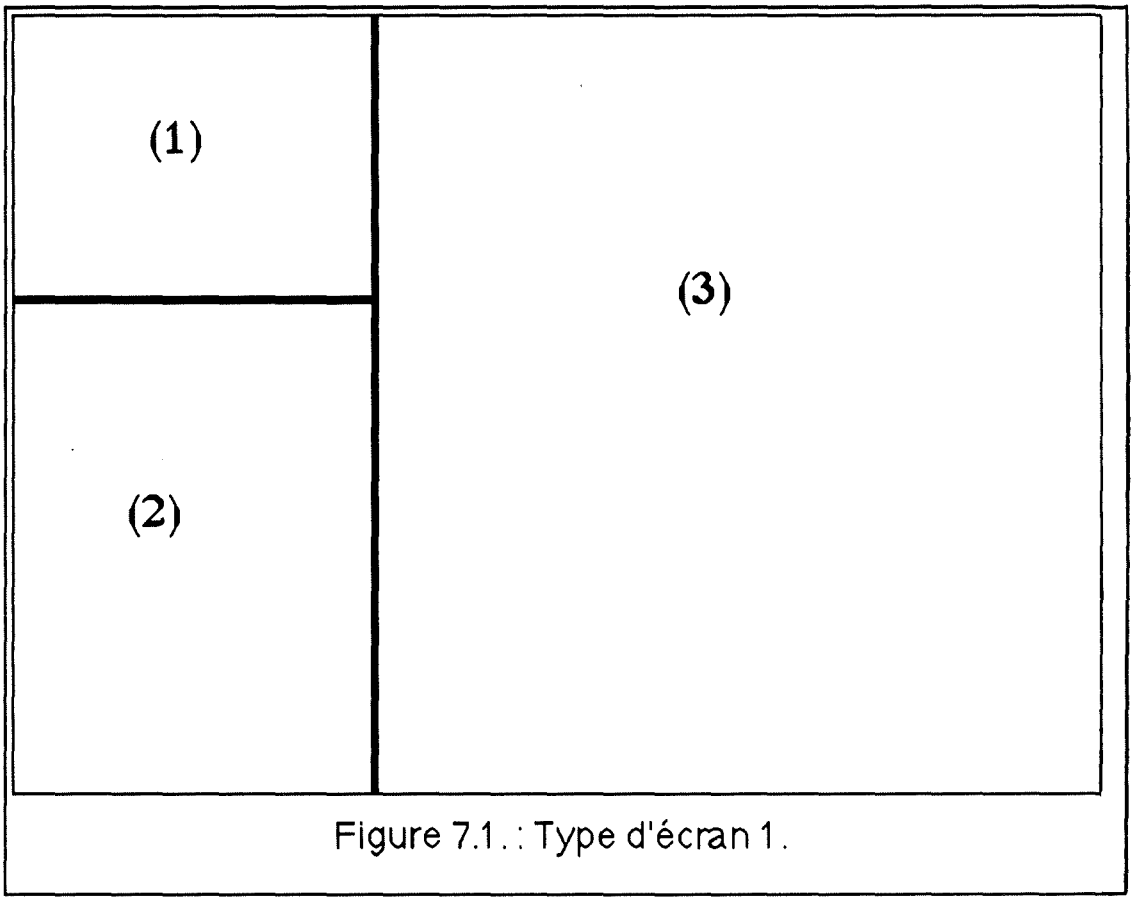


Figure 7.1. : Type d'écran 1.

Le second type d'écran (Figure 7.2.) contient, dans sa partie gauche, des boutons (icônes) et, dans sa partie droite un élément graphique à sélectionner, par exemple, une page de bande dessinée, une mise en page, ou une image.

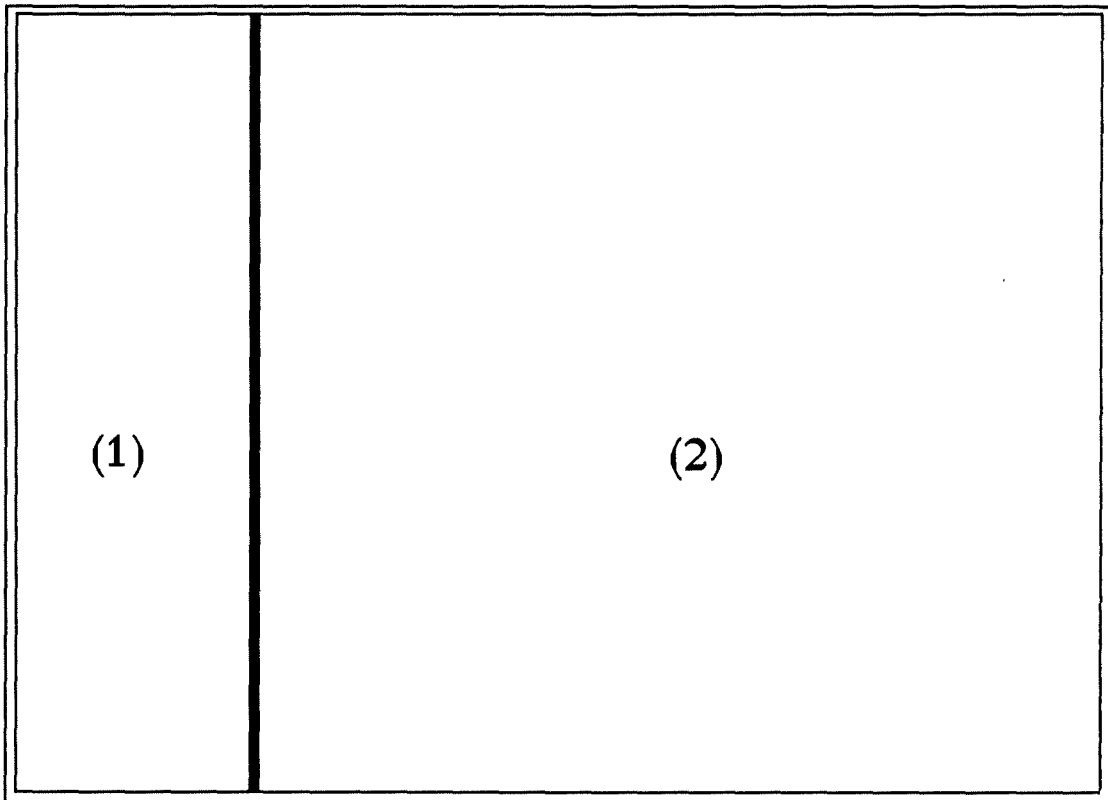


Figure 7.2. : Type d'écran 2.

Le troisième type d'écran (Figure 7.3.) offre, quant à lui, la possibilité à l'utilisateur d'effectuer un choix exclusif, en "cliquant" au moyen de la souris sur un bouton auquel sont associées des informations textuelles et graphiques, entre différentes propositions.

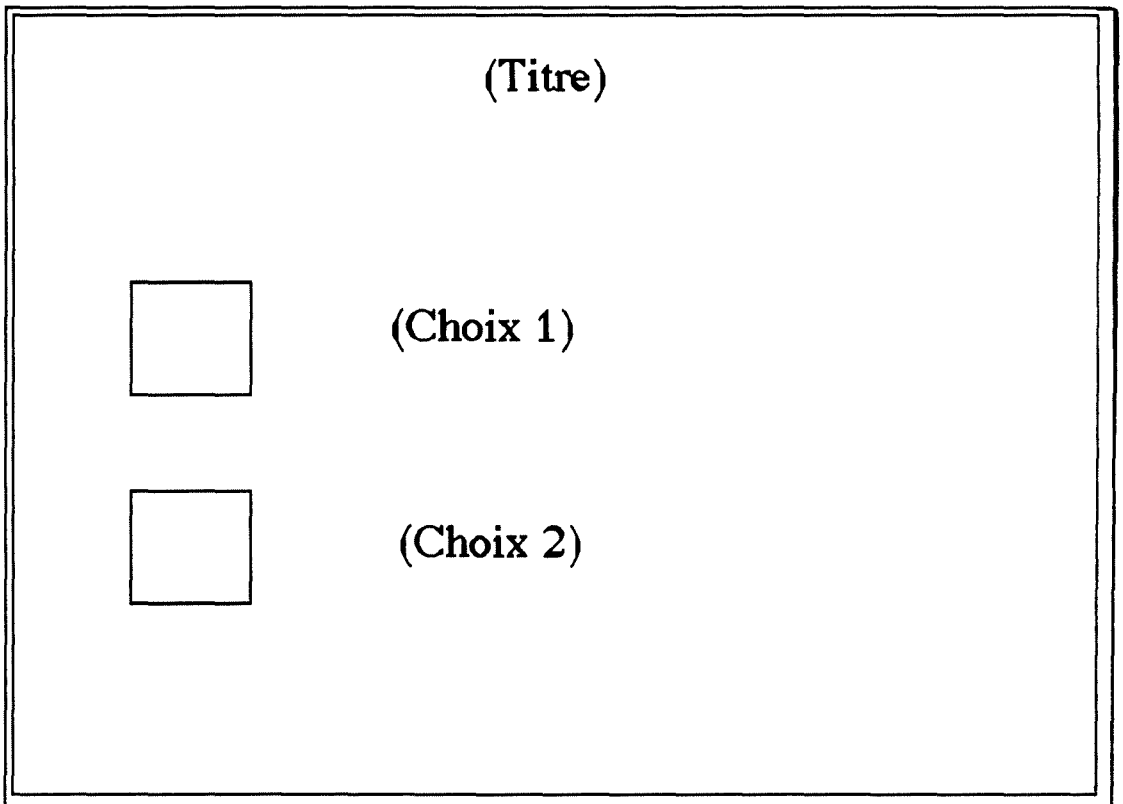


Figure 7.3. : Type d'écran 3.

Les "fenêtres de dialogue" qui sont concrétisées, dans le "Logiciel BD", par les "Fenêtre_Sélection_Texte", les "Fenêtre_Confirmation",... ont, en fonction de leur type, une représentation similaire. Ceci aide l'utilisateur à comprendre le type d'action en cours. Ainsi, la "Fenêtre_Confirmation", qui propose à l'utilisateur de confirmer un choix, comprend toujours un bouton "OK" de confirmation et un second bouton "KO" d'infirmation. Quant à la "Fenêtre_Sélection_Texte", elle permet à l'utilisateur de choisir un nom parmi une liste de noms proposés [4]. De plus, toujours dans le but de conserver une certaine cohérence, nous avons veillé à ce que ces fenêtres ne se recouvrent pas entre-elles, ou ne recouvrent pas des parties quelconques d'écrans.

Les actions de sélection, que ce soit d'une page de bande dessinée, d'une mise en page, d'une case,..., ou les actions de défilement du contenu de la "case-mémo" sont semblables. L'utilisateur peut faire défiler les objets ou passer de l'un à l'autre en utilisant des flèches qui représentent les actions "Suivant" et "Précédent".

Toujours dans le but de respecter l'homogénéité et la cohérence, toute tâche similaire a été réalisée selon une suite de commandes identiques. Nous pensons ainsi diminuer les efforts cognitifs et de mémorisation de l'utilisateur.

VII.3.1.3. La concision

Nous avons, autant que faire se peut, offert à l'utilisateur le moins possible d'actions à réaliser lorsqu'il s'agit d'effectuer une tâche. De même, nous ne lui présentons, lors d'un affichage, que les éléments utiles et nécessaires à la compréhension de la tâche en cours de réalisation.
Mais, être concis peut aussi signifier offrir la possibilité d'utiliser un raccourci, comme le fait de créer et d'employer une macro-commande. Cependant, nous estimons que cette possibilité ne peut être disponible pour une personne ayant une déficience mentale. Car, dans ce cas-ci, nous devons faire appel à la notion d'utilisateur expérimenté ou, du

[4] rappelons que ces types de fenêtre correspondent chacun à des classes appartenant au sous-système "Interface homme/machine" de la découpe orientée objet.

moins, étant capable de réaliser une abstraction et n'ayant pas de problèmes de mémorisation. Or, nous croyons que ces capacités ne sont pas celles d'une personne mentalement déficiente.

VII.3.1.4. La flexibilité

Pour respecter cette règle, l'idée est d'offrir essentiellement une interface paramétrable. Ce qui s'avèrerait être très utile, voire indispensable, vu l'hétérogénité de la population des personnes ayant un handicap intellectuel. En plus de cette possiblité de créer une "nouvelle" interface, l'utilisateur pourrait aussi avoir à sa disposition des mécanismes lui permettant de modifier certaines caractéristiques d'affichage telles que la taille des caractères, la taille et le nombre des boutons...

Cependant, nous n'avons pas implémenté une interface totalement paramétrable pour des raisons de temps et de taille de l'application. Voici toutefois une liste de paramètres que nous pensons être intéressants à implémenter pour permettre à l'animateur d'adapter l'interface de "Logiciel BD" aux besoins de l'utilisateur:
- modifier la taille des caractères.
- changer de police de caractères.
- choisir une palette de couleurs.
- afficher - ou non - la "case-mémo".
- utiliser - ou non - le code Bliss.

VII.3.1.5. Le feed-back et le guidage

Pour l'application réalisée, nous n'avons implémenté que le **feed-back visuel.** Toutefois, ce feed-back existe sous différentes formes. La première forme consiste à afficher chaque bouton sélectionné par l'utilisateur en vidéo inverse. C'est une manière d'indiquer à l'utilisateur que son action a été prise en compte. Ce procédé, appelé *"Highlighting"* en anglais, est également appliqué quand il s'agit :

- de la sélection d'une case, que ce soit dans une mise en page ou dans une page de bande dessinée; dans ce cas, le contour de cette case est mis en évidence par un encadré spécial.

- de la sélection d'un nom parmi un ensemble de noms affichés. Dès lors, c'est le rectangle (contenant le nom en question) qui est mis en vidéo inverse. Par ailleurs, le nom sélectionné est inscrit dans un autre endroit de la fenêtre.

La deuxième façon consiste à indiquer à l'utilisateur, par l'affichage d'un message [5], ce que fait le système. La plupart du temps, le message graphique ou textuel consiste en un "sablier" qui indique à l'utilisateur la présence d'un certain temps d'attente. Dans le cas d'un message graphique, un "sablier" est une icône indiquant que l'utilisateur doit attendre quelque peu et que, pendant ce temps, le système ne prend pas en considération ses actions sur le périphérique d'entrée. Le recours à un "sablier" de type graphique, en plus des messages textuels, nous semble nécessaire pour les personnes souffrant d'une déficience mentale. Car un message écrit peut ne pas fournir beaucoup d'informations à cet utilisateur. Le plus important pour ces personnes n'est pas de savoir ce que fait l'ordinateur (ce qui nécessiterait, en plus, un certain effort mental), mais d'être informé qu'il y a un temps de réponse plus ou moins long.

La troisième et dernière manière va nous permettre d'informer l'utilisateur sur son espace de travail. Ainsi, par exemple, l'utilisateur a-t'il toujours la possibilité de visualiser l'entièreté de la page de bande dessinée qui est en cours de réalisation.

Quant au **feed-back auditif**, si nous avions pu l'implémenter, il aurait consisté à indiquer de façon audible ce que l'ordinateur attendait de l'utilisateur ainsi qu'à renforcer les messages écrits. Ce type de feed-back serait en fait idéal pour accompagner les consignes données à l'utilisateur et affichées dans les fenêtres. L'utilisateur aurait pu aussi demander à l'ordinateur de "prononcer" à nouveau le message.

En ce qui concerne le **guidage**, nous avons décidé de ne pas l'offrir à l'utilisateur handicapé mental étant donné que toutes les actions à réaliser sont assez triviales et qu'elles sont supervisées par un animateur présent lors de chaque scéance de travail.

---

[5] graphique et/ou textuel

VII.3.1.6. La charge informationnelle

Afin de réduire le plus possible la charge mnésique de la mémoire à court terme, nous proposons un procédé permettant de visualiser le dessin de la page de BD qui précède celui sur lequel on travaille à un moment donné. Ceci nous a amené à afficher une fenêtre particulière, appelée **"case-mémo",** qui permet à l'utilisateur de faire défiler le contenu des cases précédentes tout en restant dans le même écran de travail. Cette "case-mémo" possède un taille relativement réduite à cause des limites des dimensions de l'écran. Nous avons alors pensé qu'il était nécessaire de concevoir un dispositif permettant d'afficher ces dessins en entier à la demande de l'utilisateur.

La représentation graphique de cette case est la suivante :

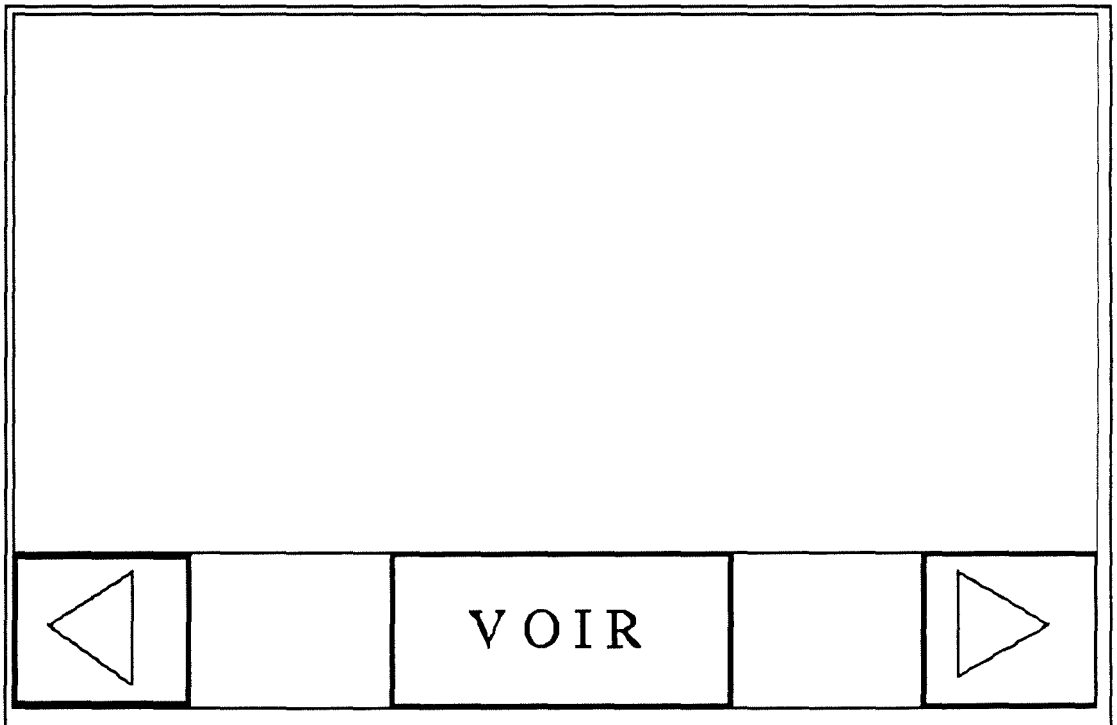


Figure 7.4. : Case Mémo.

VII.3.1.7. Le contrôle explicite

Cette règle semble tout à fait évidente pour un logiciel hautement interactif comme le "Logiciel BD" qui est conçu pour des personnes mentalement déficientes. En fait, le logiciel attend en permanence une action de l'utilisateur et n'execute que ce que demande l'utilisateur afin de ne pas perturber celui-ci ou provoquer son désintérêt.

VII.3.1.8. La prévention et la gestion des erreurs

Plusieurs mécanismes pour empêcher l'utilisateur de commettre des erreurs ont été mis en oeuvre. Ceux-ci sont le "highlighting", les "Fenêtres de confirmation" et les menus.

Le rôle du highlighting est de mettre en évidence le choix de l'utilisateur qui, par conséquent, a la possibilité de confirmer son choix.

La "fenêtre de confirmation" est, comme nous l'avons vu, une fenêtre dans laquelle va se faire la confirmation d'une action. Cette fenêtre est utilisée avant chaque action importante telle que, par exemple, la destruction d'une page de bande dessinée existante.

Quant aux menus, ils sont affichés en ne contenant que les commandes utiles au contexte actuel de travail. Ainsi, nous évitons de rendre disponibles des commandes qui pourraient causer des désagréments si elles étaient utilisées par erreur. De plus, tous les menus sont organisés en réseau.

Du point de vue de la gestion, il existe deux types d'erreurs: celle résultant d'une incohérence avec le programme, et celle "voulue" par l'utilisateur. Pour le premier type d'erreurs, nous pouvons avoir, par exemple, l'introduction d'un nom d'utilisateur inexistant. Le deuxième type d'erreurs ne recouvre pas des erreurs en tant que telles. Nous avons, en fait, affaire soit à un désir de la part de l'utilisateur de revenir à l'écran précédent, soit à l'intention de l'utilisateur de "défaire" la dernière action réalisée. Pour résoudre le premier cas, nous offrons une commande équivalant à "SORTIR" et, pour le second, l'utilisateur dispose d'une

commande réalisant un "UNDO" (défaire les effets de la dernière action réalisée par un "outil").

Dans certains cas, l'erreur doit être signalée à l'attention de l'utilisateur. Cet avertissement se fait de façon homogène en affichant un message dans une zone de l'écran prévue à cet effet.

VII.3.1.9. Le séquencement des actions

Toute action réalisée avec le "Logiciel BD" est organisée en une séquence ayant un début et une fin. Ainsi, la création d'une nouvelle page de Bande Dessinée commence par la sélection ou la création d'une mise en page, se poursuit par la création des différents dessins dans cette page, et se termine lorsque l'on active le bouton permettant de retourner au point de départ.

VII.3.2. Le style de dialogue mis en oeuvre

Comme il l'est conseillé dans la partie concernant les aspects théoriques de l'interface homme/machine dans ce chapitre, le type de dialogue mis en oeuvre consiste, bien souvent, en l'implémentation de deux, voire trois styles de dialogue [6].
Dans le cas de "Logiciel BD", nous avons implémenté un panaché qui se compose principalement de deux styles de dialogue: les menus et la manipulation directe.

Il existe dans "Logiciel BD" différents types de menus qui contiennent tous plusieurs boutons. Le premier type de menu consiste à offrir à l'utilisateur le choix entre diverses propositions formulées textuellement et accompagnées d'un bouton représentant graphiquement cette proposition. Le second propose, lors de la création d'un dessin ou de la manipulation d'une image, une série d'outils disponibles et représentés par des boutons rangés sous la forme d'une matrice. Le troisième propose quelques boutons alignés verticalement à gauche de l'écran, comme c'est

---

[6] ce que nous appelons le panaché.

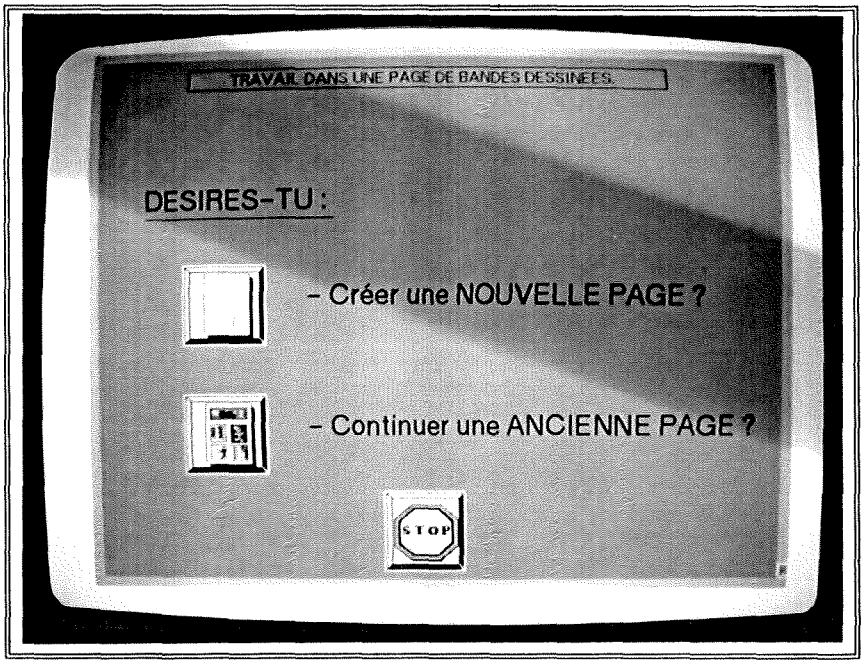
par exemple le cas pour l'affichage d'une page de Bande Dessinée en entier.

Ces menus qui s'enchaînent selon un réseau, se distinguent par leur organisation et la localisation à l'écran des éléments qui les composent. De plus, nous ne présentons qu'un seul menu à la fois. Quant aux boutons, ils peuvent être mutuellement exclusifs (on ne peut donc en sélectionner qu'un seul à la fois). En ce qui concerne la sélection des boutons dans les menus, elle s'effectue au moyen de la souris qui est le principal "Direct Input Device" pris en considération par l'application. Le deuxième est le clavier dont l'utilisation se résume à l'introduction de texte, telle que les saisies des références de l'utilisateur ou l'introduction de texte dans les "Bulles" ou "Commentaires".

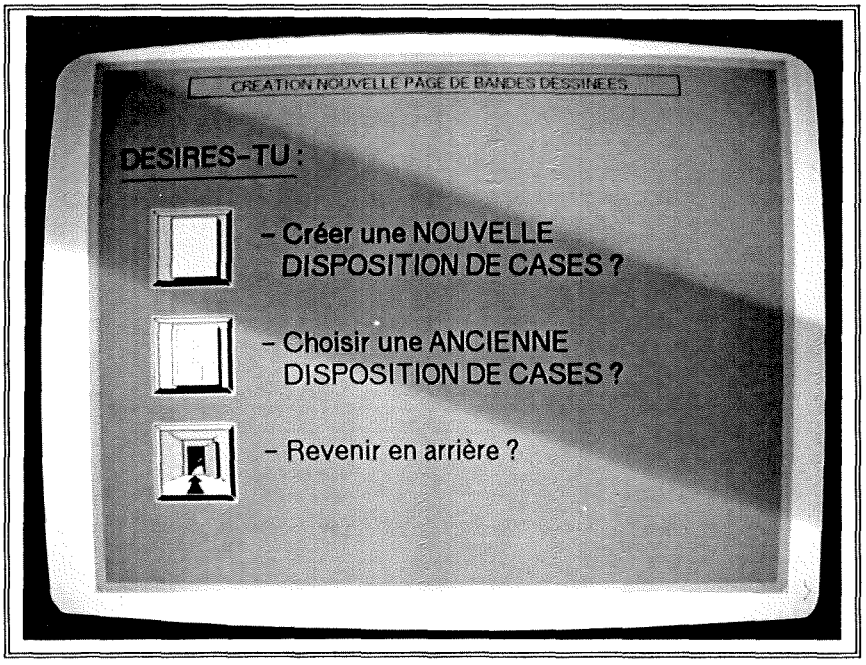
Nous utilisons la manipulation directe quand il s'agit de la création d'un dessin, et plus particulièrement lorsque l'on effectue des retouches à ce dessin telles que le gommage, le crayonnage le déplacement et le collage d'une image dans ce même dessin.

VII.3.3. La couleur

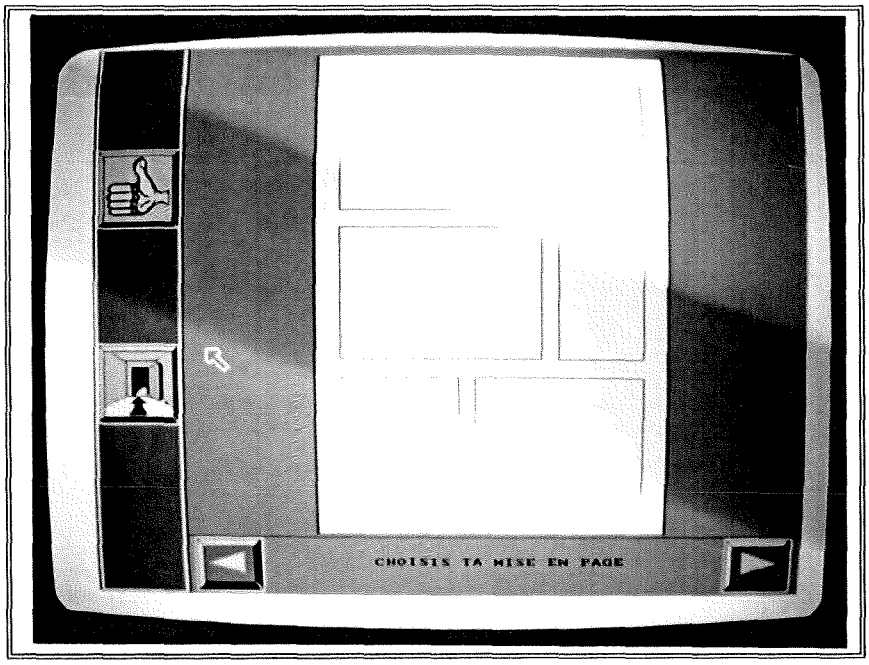
Bien qu'étant conscients de l'importance que peut jouer l'utilisation de couleurs dans l'interface utilisateur du logiciel, nous avons été contraints, pour des raisons techniques d'implémentation liées à la technologie de l'ordinateur utilisé, de concevoir une interface en noir et blanc. Il s'agit, en fait, d'une interface qui utilise 16 couleurs qui sont uniquement des nuances de gris.
Il fallait, en effet, permettre l'utilisation simultanée de plusieurs images possédant chacune leur propre palette de couleurs. Or, dans l'application, l'interface partage la même palette de couleurs que les images manipulées par le logiciel. Afin de conserver toujours la même palette de couleurs pour l'interface, il faut donc adapter la palette de chaque nouvelle image utilisée par le logiciel à celle de l'interface. Cette modification peut rendre les images totalement méconnaissables par rapport à leur état original lorsqu'on n'utilise pas une palette de couleurs "grisées". La "destruction" de l'image qui s'ensuit est encore accentuée, dans notre cas, par le fait du



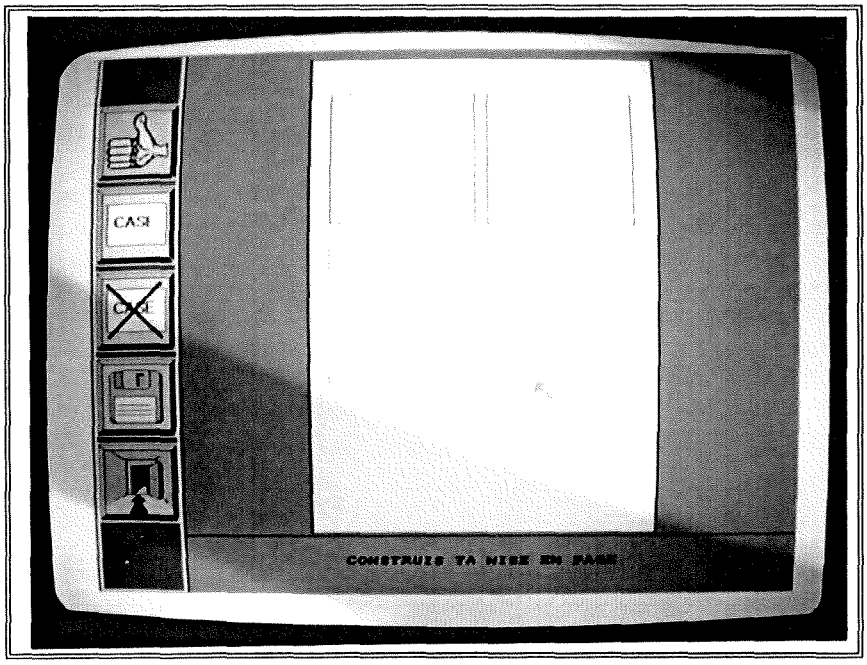
Ecran 1: Choix du travail avec une nouvelle ou une ancienne page de bandes dessinées.



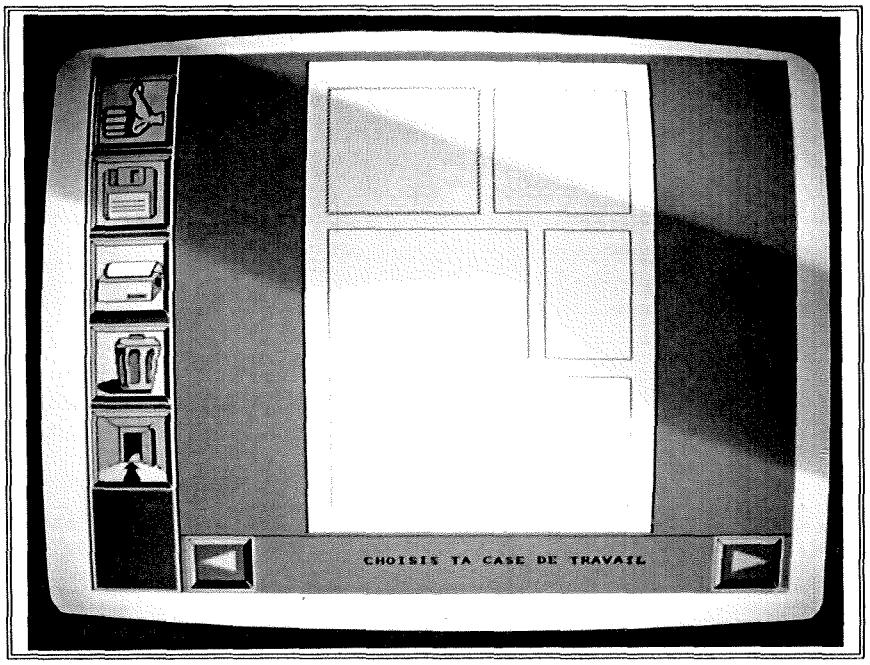
Ecran 2: Choix de la création d'une nouvelle mise en page ou de l'utilisation d'une ancienne.



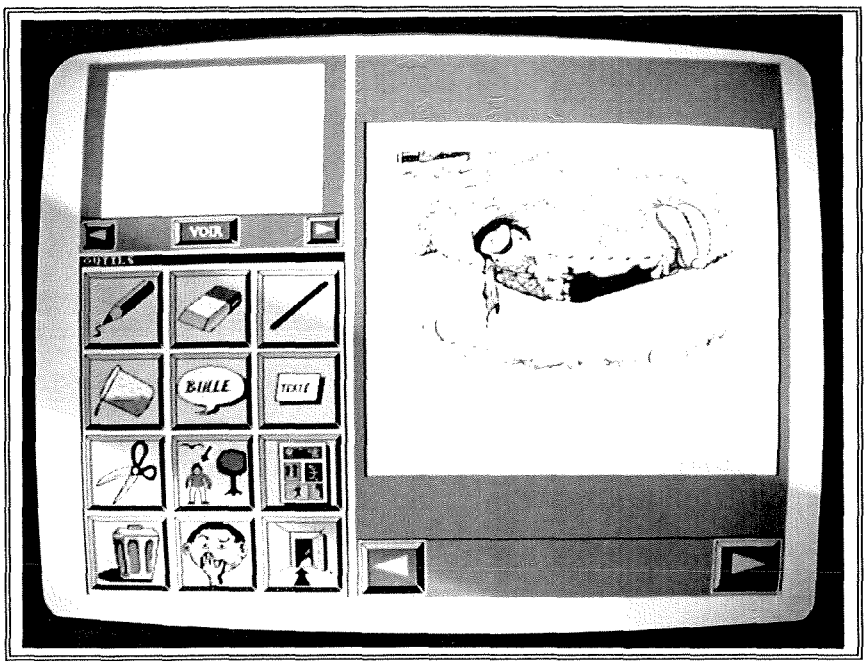
Ecran 3: Choix d'une ancienne mise en page.



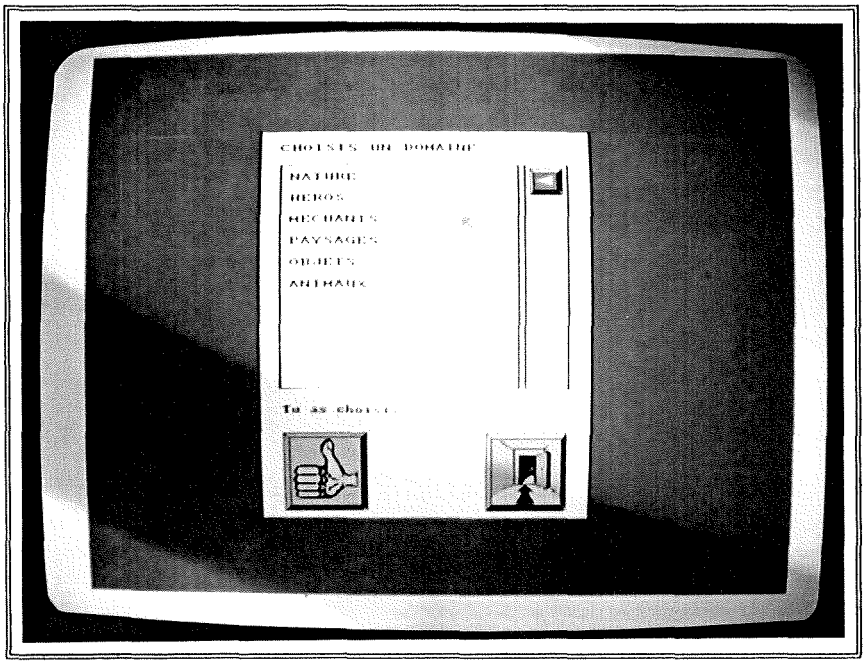
Ecran 4: Création d'une nouvelle mise en page.



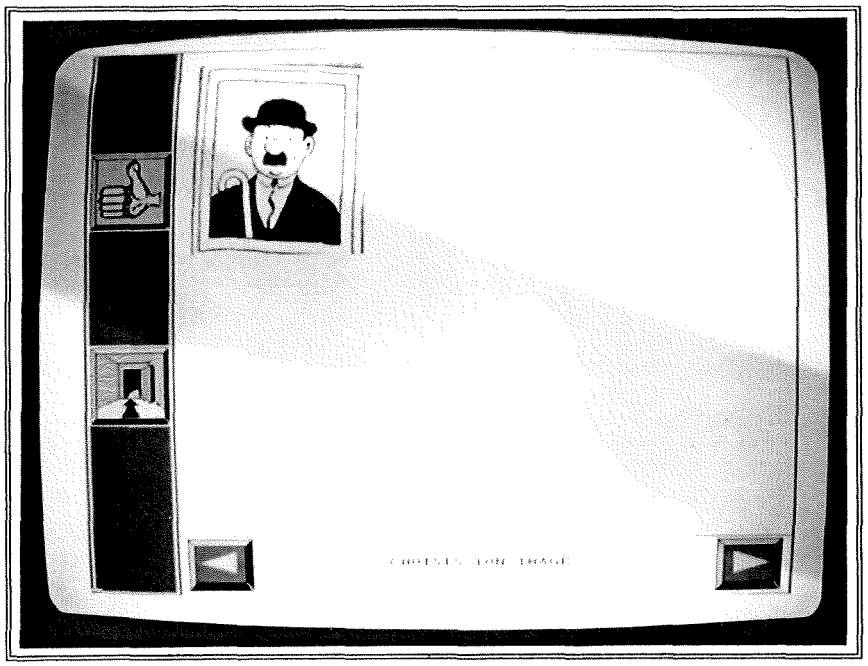
Ecran 5: Choix d'une case de travail à partir de la page de BD.



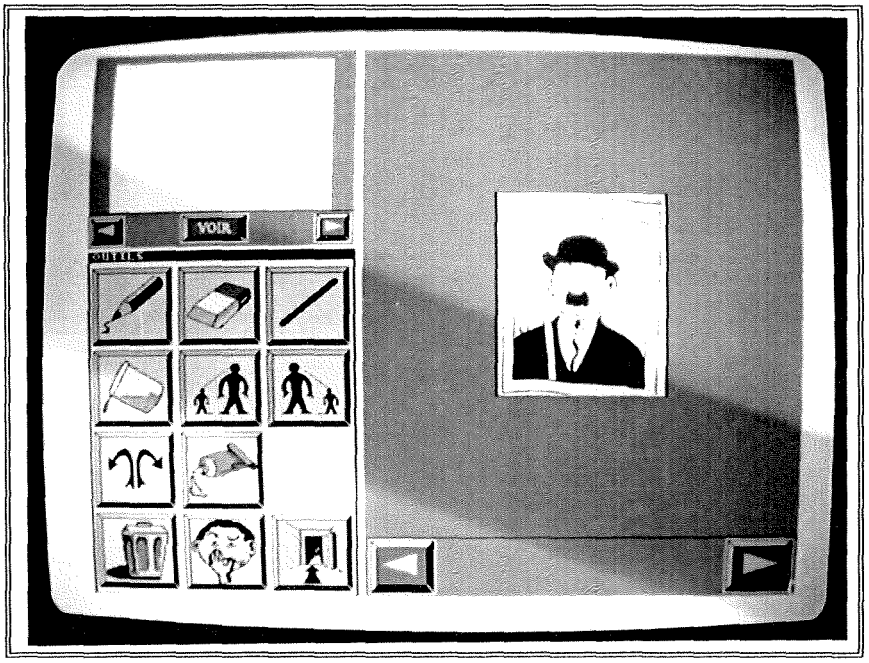
Ecran 6: Outils de travail dans le dessin d'une case.



Ecran 7: Choix d'un domaine dans la banque d'images secondaire.



Ecran 8: Choix d'une image dans la banque d'images secondaire.



Ecran 9: Outils de travail avec une image active.

nombre fort restreint de couleurs affichables simultanément à l'écran imposé par la machine utilisée [7].
Il a donc été nécessaire d'avoir recours à l'utilisation de 16 nuances de gris pour éviter la dégradation de ces images. Ce choix n'a cependant pas été pénalisant en ce qui concerne l'esthétique de l'interface et des images manipulées par le logiciel. Il est même possible de manipuler des images digitalisées qui parviennent à conserver leur aspect "réaliste", lequel aurait été perdu par l'utilisation de la palette restreinte (16 couleurs) de couleurs non-grises.

Bien que de plus en plus utilisées, les couleurs (non-grises) ne sont pas toujours obligatoires et peuvent être parfois remplacées par des nuances de gris sans provoquer une réelle perte de convivialité pour l'utilisateur. "La couleur n'est pas obligatoirement le *nec plus ultra*: les écrans monochromes de NeXT, avec une très haute définition et un dessin d'interface confié à un designer, rivalisent sans difficulté avec les autres. La couleur demeure l'affaire des graphistes." [MICSS, 90]

VII.3.4. Le prototype de "Logiciel BD"

En ce qui concerne le prototypage du logiciel, nous nous sommes limités à la présentation d'un prototype constitué des différents écrans enchaînés. Il s'agissait d'un parcours structuré d'écrans et d'actions sur certains éléments de l'interface, comme l'enfoncement de différents boutons. Après une première évaluation du prototype, nous avons pu constatater que l'interface du logiciel est relativement bien acceptée par la population qui l'a testée. A savoir, les personnes mentalement déficientes participant aux ateliers du centre PSINHA des F.N.D.P. de Namur. Nous pensons qu'il est important de souligner que ces personnes possédaient au préalable une expérience sur micro-ordinateur, notamment avec l'APPLE II GS et son logiciel de dessin GS PAINT. La signification des représentation graphiques liées aux boutons fut aussi facilement comprise.
Ce prototypage nous a aussi confirmé qu'il fallait utiliser les deux boutons de la souris en les assigant à des actions identiques.

---

[7] à savoir l'Amiga 2000 de Commodore.

Lors de l'exécution des spécifications de l'interface, nous nous sommes demandés, par exemples, si le contenu des messages était judicieux, si la définition des données était suffisamment précise, ou encore si il n'y avait pas d'informations manquantes ou redondantes. Par cette approche, nous avons favorisé la participation active des utilisateurs à la conception du système d'information et de l'évaluation de leurs spécifications.

VII.3.5. Le cahier des charges de "Logiciel BD"

Le contenu de notre cahier des charges se base, entre autres, sur un modèle essentiellement empirique de l'utilisateur final. Pour ce faire, nous nous sommes bien sûr basés sur les connaissances qu'ont les psychologues et éducateurs de cette population, mais également sur notre démarche auprès de trois institutions, à savoir:
- le PSINHA (PSychologie, INformatique et HAndicap) des Facultés universitaires Notre-Dame de la Paix à Namur, qui est un projet orienté sur les aspects psychologiques et médicaux de la réadaptation de personnes handicapées par le biais de l'informatique.
- les CREAHM (CREAtivité Handicap Mental) de Bruxelles et de Liège, qui organisent des ateliers créatifs dans différentes disciplines pour les personnes mentalement déficientes.

VII.4. CONCLUSIONS

Nous constatons que, pour réaliser une interface adaptée aux besoins de futurs utilisateurs d'un logiciel, plusieurs éléments interviennent dans la conception de celle-ci.
Les deux éléments essentiels qu'il faut déterminer avant de pouvoir commencer la conception d'une interface homme/machine, sont les tâches que le logiciel doit absolument réaliser et les utilisateurs pour lesquels il est conçu. Nous dirons donc que l'analyse de ces deux entités est le **pré-requis** à toute conception d'interface.
C'est seulement lorsqu'on connaît ces deux éléments qu'il est permi de commencer la conception de l'interface adaptée à la population cible. Sa construction peut être bâtie à partir des 9 règles empiriques qui jouent un

rôle dans cette conception, aussi bien du point de vue fonctionnel que du point de vue du dialogue. Il faut, de plus, construire une interface respectant, à divers degrés, les cinq critères de qualité, permettant ainsi d'obtenir, normalement, l'interface la plus conviviale possible.

L'interface, une fois construite, met en oeuvre divers écrans et style(s) de dialogue au travers desquels se réalisera le dialogue entre l'utilisateur et l'ordinateur. Chaque écran doit être conçu de manière à ne pas perturber l'utilisateur, notamment en utilisant des icônes appropriées. L'évaluation de l'interface peut être réalisée grâce à un prototype qui, dans notre cas, s'est limité au dialogue et aux écrans. Le cahier des charges qui contient les fonctionnalités et le dialogue a, quant à lui, été étudié avec les psychologues.

Cependant, si tout logiciel doit passer par une phase d'évaluation pour voir si il répond bien aux attentes de l'utilisateur, "la mise en oeuvre de l'ergonomie devrait ... se faire à chaque étape de la conception, le plus tôt possible, plutôt qu'a postériori, lors d'une contribution évaluative extérieure" [SCAPI,86].

Chapitre VIII

Aspects graphiques du logiciel

VIII.1. INTRODUCTION

Ce chapitre propose quelques-uns des divers concepts concernant le graphisme en informatique avec lesquels il a été nécessaire de se familiariser avant de réaliser la conception et l'implémentation de "Logiciel BD". Dans un premier temps, nous avons dû chercher à savoir comment sont gérées les images par la plupart des ordinateurs personnels et plus particulièrement par l'Amiga. Ensuite nous nous sommes intéressés à la façon dont ces ordinateurs permettent la réalisation de dessins. C'est ainsi que nous présentons les deux grands types de dessin gérés par les machines actuelles; les dessin vectorisé et le dessin point par point. Le troisisème concept graphique qui nous a préoccupé est celui qui concerne les transformations d'images graphiques. Nous n'aborderons dans ce chapitre que le début de la théorie des transformations qui est à la base de bon nombre d'algorithmes de manipulations d'images. L'utilisation de ce type de transformations convient cependant mieux au dessin vectorisé qu'au dessin point par point. Finalement, c'est sur les caractéristiques graphiques propres à l'Amiga que nous avons dû nous pencher afin d'analyser la manière dont il gère le graphisme. Nous aborderons ainsi le

problème du tremblement de l'image d'un écran en mode entrelacé sur l'Amiga ainsi que le transfert de fichiers d'images graphiques entre applications et entre ordinateurs.

VIII.2. ORGANISATION D'UNE IMAGE SUR ORDINATEUR

Nous allons voir dans ce point comment est organisée, de façon logique et physique, une image sur Amiga. Cette organisation est à rapprocher de celles qui sont utilisées sur la plupart des ordinateurs actuels. C'est la raison pour laquelle nous n'avons pas hésité à la présenter dans ce mémoire.

VIII.2.1. Représentation logique d'une image

Une image peut être vue comme étant une matrice à deux dimensions de points appelés pixels. Elle est caractérisée par le nombre de points, en abcisse et en ordonnée, qui la constituent. Ainsi l'Amiga peut afficher un écran constitué de 640 points en abcisse et 512 points en ordonnée, et cela en haute résolution et en mode "entrelacé". On dira qu'il affiche une résolution 640x512.

Chaque point de la matrice possède sa propre couleur. Cette couleur appartient obligatoirement à la palette de couleurs fixée pour l'écran à un moment donné. Une palette de couleurs est en fait une table à une dimension dont chaque élément correspond à une couleur. La taille de cette table est fonction du nombre de couleurs simultanées que peut afficher un écran à un moment donné[1]. Ainsi, chaque couleur que l'on peut afficher possède un numéro que l'on peut utiliser comme indice dans la table pour retrouver la caractéristique de cette couleur. Comme on le voit, le nombre maximal de couleurs affichables en même temps à l'écran[2] est fonction de la taille de codage de l'index de chaque couleur dans la palette. Ainsi, si l'index est codé sur 5 bits de longueur, il sera possible d'afficher 32 (2^5) couleurs à la fois.

---

[1] sa taille peut cependant dépasser ce nombre dans certains cas.

[2] correspond au nombre de couleurs dans la palette.

Une image étant composée de points et chaque point étant caractérisé par une couleur appartenant à la palette de couleurs utilisée, on peut voir une image comme étant le rassemblement d'une série de valeurs d'indice dans la palette de couleurs, chacune de ces valeurs d'indice correspondant à un pixel de l'image. Donc, si l'index est codé sur 5 bits, l'image sera représentée par un ensemble de groupes de 5bits.

Comme on le voit à la figure suivante, une image est organisée logiquement en bitplanes (plans de bits). Le bitplane i est constituée de l'ensemble des bits d'ordre i de la représentation binaire du numéro d'indice de la couleur des pixels constituant l'image. Pour retrouver le numéro d'indice de la couleur d'un point, il faut prendre les bits correspondant à ce point dans chaque bitplane et reconstituer l'indice comme expliqué dans la figure.

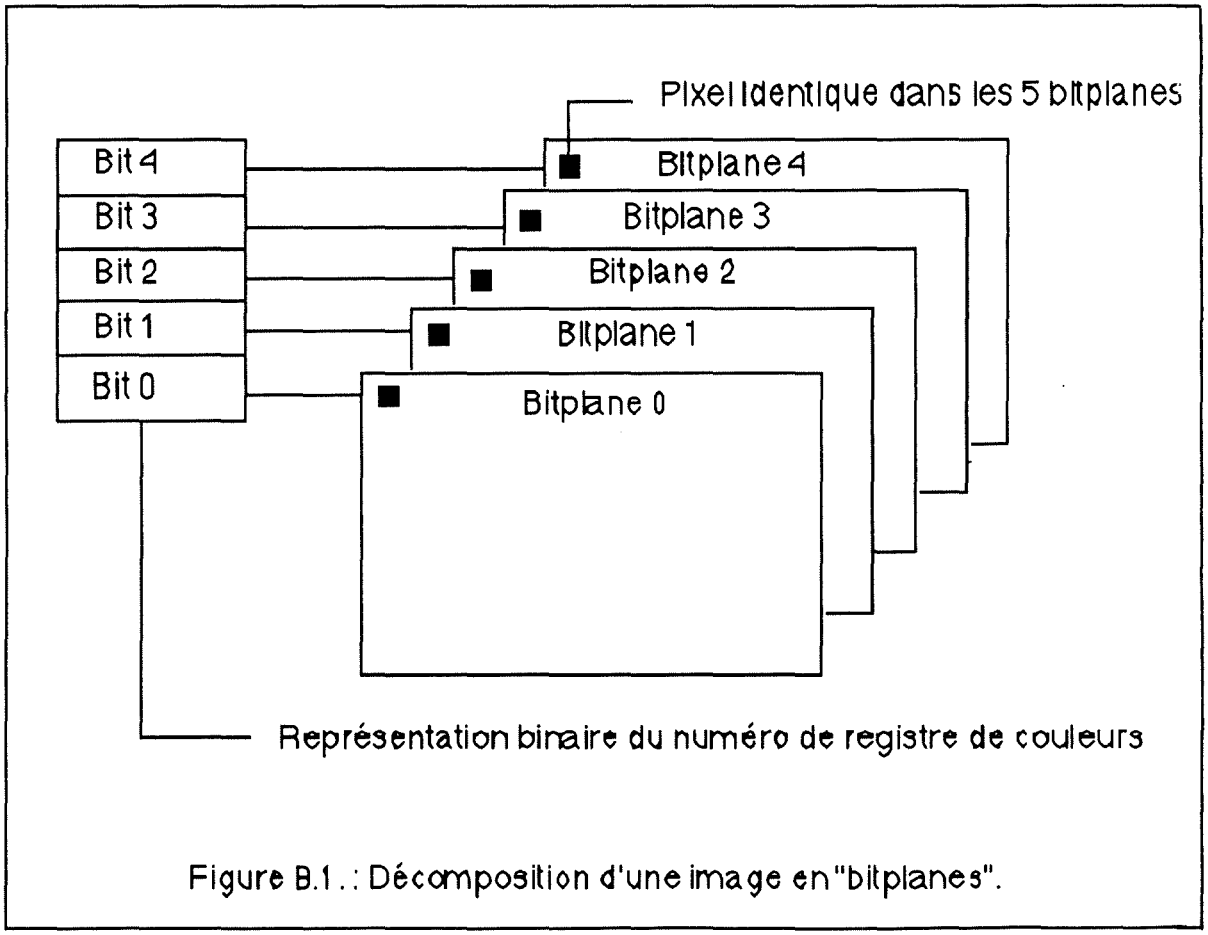


Figure B.1. : Décomposition d'une image en "bitplanes".

On peut ajouter que ce n'est pas parceque cinq bitplanes ont été définis pour toutes les images d'un écran donné que toutes les images appartenant à cet écran doivent avoir 32 couleurs. 32 (couleurs) est simplement le nombre maximal de couleurs que peuvent avoir ces images. Des images qui n'auraient que deux couleurs mais qui seraient définies pour un écran composé de cinq bitplanes (32 couleurs) seraient aussi composées de cinq bitplanes.

Si on ne défini qu'un seul bitplane pour l'écran, le nombre de couleurs maximal de la palette serait toujours de 2 (2^1), par contre, la taille des images définies pour cet écran serait réduite par rapport à celles définies pour un écran de 5 bitplanes.

VIII.2.2. Représentation physique d'une image

L'organisation physique des données d'une image suit la représentation logique de celle-ci. Sur l'Amiga, les informations seront codées en mémoire dans des blocs consécutifs de 16 bits[3] . On trouvera, d'abord toutes les informations du bitplane 0 suivies de celles du bitplane 1 et ainsi de suite. A l'intérieur d'un bitplane, les informations sont prises en ligne et ensuite en colonne.

VIII.3. LE DESSIN POINT PAR POINT ET LE DESSIN VECTORIEL

Il existe deux grandes façons de dessiner par ordinateur. La première concerne de dessin point par point dans la mémoire de l'écran, la seconde consiste à travailler en mode vectoriel. Nous présentons ces deux approches dans les points qui suivent bien que nous n'ayons retenu que celle qui concerne le dessin point par point pour l'implémentation de l'application de "Logiciel BD". La justification de ce choix peut se trouver dans le fait que nous avons été obligés de travailler avec des images digitalisées dans le logiciel.

---

[3] correspond à la longueur d'un mot sur Amiga.

VIII.3.1. Le Dessin point par point

Le dessin point par point revient à modifier l'aspect de l'écran en altérant le contenu de la mémoire écran[4] et ce, pixel par pixel. C'est de cette manière qu'opèrent toutes les primitives de dessin offertes par quelque système graphique que ce soit. Ainsi, lorsqu'on demande à la fonction *rectangle()* d'en tracer un aux coordonnées indiquées, elle procède en modifiant un à un tous les points définissant l'aspect visuel de cet objet. On dira que ces primitives de dessin travaillent en mode **point par point.**

Dans cet esprit, tracer une ligne de coordonnées (0,0) à (0,10) revient à dessiner un à un 11 points contigus verticalement.

Le point faible du dessin point par point est que le résulat obtenu dépend du périphérique utilisé. Ainsi, s'il est aisé de distinguer la ligne de 11 points définie plus haut sur un écran de résolution 320x256 de l'Amiga, nous aurons plus de difficultés à l'apercevoir sur une page sortant d'une imprimante dont la résolution serait de 300 DPI (Dots Per Inch). Un simple calcul montre en effet que 11 points sur un tel périphérique mesureront moins d'un millimètre ((2,54 / 300) * 100) = 0,93133) de longueur.

Pour solutionner ce problème des résolutions variables, il faudrait changer les coordonnées des objets tracés, et cela en fonction de la précision du support de dessin utilisé (écran, imprimante, table traçante). Ainsi, pour représenter un objet (ligne, rectangle, cercle, ...) d'abord sur un périphérique à 100 DPI, puis sur un autre à 300 DPI tout en lui conservant ses proportions, il suffirait de multiplier ses coordonnées par trois. Toutefois, on ne peut pas faire précéder chaque instruction de dessin d'une fonction d'adaptation des coordonnées prenant en compte le périphérique de tracé courant. C'est là qu'entre en compte le dessin vectorisé.

---

4 tel que défini au point précédent.

VIII.3.2. Le Dessin vectorisé

Le principe de la vectorisation graphique consiste à intercaler entre toute application graphique et le dessin point par point une couche logicielle transparente dont le but est de reproduire toute image générée par la première à n'importe quelle échelle, et ce, dans le but de conserver l'aspect et les proportions, quel que soit son résultat final.

Les principes du dessin vectorisé sont les suivants:
Alors qu'en mode point par point, on considère la ligne qui relie les coordonnées (0,0) à (0,10) comme un ensemble de 11 points, en mode vectoriel, on lui conserve ses caractéristiques initiales, de telle sorte qu'elle reste une ligne dont les coordonnées terminales sont respectivement (0,0) et (0,10). De cette façon, chaque objet créé par l'application est défini par ses caractéristiques minimum. Pour une ligne, ce sont ses deux points terminaux, pour un rectangle, ce sont les deux points d'une de ses diagonales, etc. Lors du tracé final en mode point par point, il suffit d'adapter les coordonnées de l'objet vectoriel à la résolution du périphérique courant pour obtenir un résultat irréprochable.

VIII.4. Transformations d'images

Le "Logiciel BD" est principalement un logiciel de manipulation d'images graphiques à deux dimensions; c'est la raison pour laquelle il est intéressant de présenter les principes de transformation d'images sur lesquels se basent la plupart des logiciels de création graphique actuels. Ces principes ne sont toutefois intéressants et performants que pour le travail avec des formes géométriques prédéfinies dont les points caractéristiques[5] sont connus.

---

[5] ensemble minimum des points à partir desquels il est aisé de reconstruire la forme géométrique désirée.

VIII.4.1. Principes de transformation

"Deux aspects de la formulation des transformations doivent être pris en compte:

1. Une transformation est une unité mathématique unique et par conséquent peut être notée par un nom ou un symbole unique:

2. Deux transformations peuvent être combinées, ou *concaténées,* pour former une transformation unique qui a le même effet que l'application séquentielle des deux originales. Ainsi, la transformation *A* peut être une translation et la transformation *B* un changement d'échelle. La propriété de concaténation nous permet de déterminer une transformation $C = AB$ dont l'effet est de translater puis de changer d'échelle." [WILL,79]

Chacune de ces transformations est utilisée en vue de générer un nouveau point (x',y') à partir des coordonnées d'un point (x,y) dans la description de l'image originale. Si cette définition originale comporte une ligne, il suffit d'appliquer la transformation aux points extrêmes de la ligne et de tracer une ligne entre les deux points extrêmes transformés.

Quelques transformations:

La translation

La forme d'une translation est la suivante:

$$x' = x + Tx \qquad y' = y + Ty$$

La rotation

Pour faire pivoter un point (x,y), dans le sens des aiguilles d'une montre, en fonction d'un angle Teta à partir de l'origine du système de coordonnées, on écrit:

$$x' = x \cos Teta + y \sin Teta \qquad y' = - x \sin Teta + y \cos Teta$$

Le changement d'échelle

Les transformations d'échelle sont:

$$x' = x\,Sx \qquad y' = y\,Sy$$

VIII.4.2. Concaténation de transformations

Comme nous l'avons vu, des séquences de transformation peuvent être combinées pour former une seule transformation grâce au mécanisme de *concaténation.* Il est, en effet, rare que l'on veuille appliquer une seule transformation à la fois, telle que par exemple une rotation ou un changement d'échelle par rapport à l'origine. On a souvent besoin de réaliser des transformations plus complexes comme par exemple une rotation autours d'un point quelconque. Une rotation autours d'un point quelconque peut être réalisée en appliquant une séquence de trois transformations simples: une translation, suivie d'une rotation, suivie d'une autre translation.

L'utilisation de transformations concaténées possède différents avantages. On peut les représenter de façon plus compacte que pour une séquence, et on peut généralement calculer la transformation avec moins d'opérations arithmétiques que si on avait à appliquer chacune des transformations en séquence les unes après les autres. Cependant les règles pour concaténer des equations de transformations sont relativement complexes. Elle peuvent toutefois être simplifiées par l'utilisation de matrices pour définir ces transformations.

VIII.5. LE GRAPHISME ET L'AMIGA

L'Amiga a souvent été considéré, dans le passé, comme étant une machine qui offrait des avantages non-négligeables par rapport aux autres ordinateurs personnels en ce qui concerne le graphisme. Nous pouvons dire que ce n'est plus vraiment le cas actuellement alors que d'autres ordinateurs personnels offrant des caractéristiques identiques, voire supérieures, envahissent le marché. Nous voyons, dans ce point, quels sont

les grands modes graphiques dont dispose l'Amiga ainsi que le problème du tremblement d'images qui est lié à la technologie de l'Amiga.

VIII.5.1. Les modes graphiques de l'Amiga 2000

Les écrans peuvent avoir une basse résolution, **"lo-res"**, qui est de 320 pixels de largeur ou une haute résolution, **"hi-res"**, qui est de 640 pixels de large. Avec ces deux types de résolution, il est possible de travailler en mode entrelacé qui est de 400 pixels de hauteur ou en mode normal qui est de 200 pixels de hauteur.

Le mode **Genlock Video** signifie qu'à la place de la couleur de fond, on peut avoir un signal vidéo, fourni par enregistreur vidéo ou caméra, comme "toile de fond".

Le mode **HAM** (Hold And Modify) permet d'afficher les 4096 couleurs de l'Amiga en même temps à l'écran. Ce mode est cependant fort difficile à programmer à cause de la complexité de la technique utilisée.

Le mode **Extra halfbrite** permet d'afficher jusqu'à 64 couleurs en même temps à la place des 32 habituelles.

Disposant de caractéristiques graphiques modérément intéressantes, l'Amiga semble être, à première vue, un ordinateur approprié pour la réalisation de "logiciel BD". Nous avons cependant été confrontés à quelques problèmes de taille qui étaient dûs principalement aux caractéristiques physiques de l'Amiga 2000.

Le mode entrelacé de l'Amiga est à la source d'un problème non-négligeable si l'on est attentif à la réalisation d'un logiciel ergonomique pour personnes mentalement déficientes. Lorsque nous utilisons ce mode, ce qui est affiché à l'écran commence à trembloter (il s'agit du "flickering").
Ce tremblement peut être expliqué de la façon suivante: comme nous l'avons vu, il y a deux modes primaires d'opération dans le monde vidéo de l'Amiga, le mode entrelacé et le mode non-entrelacé. L'Amiga envoie de l'information vidéo dans des "champs" de lignes de balayage horizontal

pour réaliser un affichage sur le moniteur. En mode entrelacé, il y a deux champs, appelés champ pair et champ impair, qui pris ensembles composent un cadre vidéo complet. Ces champs sont décalés l'un par rapport à l'autre à l'écran de sorte que leur lignes de balayage respectives soient entrelacées. En d'autres mots, les lignes du champ pair alternent verticalement avec les lignes du champ impair. De haut vers le bas de l'écran, il y a une ligne du champ pair, ensuite une ligne du champ impair, une ligne du champ pair, etc., d'où le terme "entrelacement".
Le taux d'affichage vidéo des cadres vidéo est de 30 cadres par seconde. Ainsi, chaque champ (un demi-cadre vidéo) prend 1/60ème de seconde pour être dessiné à l'écran[6]. Durant le temps qu'un rayon électronique prend pour dessiner le second des deux champs séparés du cadre entrelacé actuel, le premier champs a perdu sa brillance, puisque l'éclat du point de phosphore que le rayon balaye commence à faiblir dès que le rayon de balayage passe au pixel suivant. Lorsque le taux de "rafraîchissement"[7] est trop bas - 30 Hz ou moins - pour que l'oeil humain ne voie le tout comme une seule image stable, le résultat perçu est un tremblement ennuyeux du mode entrelacé. Les écrans non-entrelacés sont en fait le MEME champ balayé DEUX FOIS pour chaque cadre d'écran, soit soixante fois par seconde, ce qui est suffisament rapide pour éviter le tremblement.
La solution trouvée pour éviter ce tremblement inacceptable a été de se procurer une carte "anti-tremblement"[8] qui nécessite l'utilisation d'un moniteur multi-fréquences (Multiscan Monitor). Ceci a, bien entendu, engendré un accroissement du coût du matériel de base nécessaire au bon fonctionnement du logiciel.

VIII.5.2. L'échange de fichiers d'image et le format IFF-ILBM

En ce qui concerne l'échange d'images entre des logiciels graphiques commerciaux, le problème est d'utiliser des fichiers images qui soient codés dans un format "compréhensible" par tous ces logiciels à la fois. Une solution est d'utiliser des fichiers codés selon le standart IFF-ILBM qui est reconnu par la grande majorité des logiciels graphiques commercialisés actuellement.

---

6 le hardware standard de l'Amiga 2000 n'est pas capable d'envoyer des données vidéo plus rapidement que cela.
7 le nombre de fois par seconde que chaque champ est redessiné.
8 A2320 Diplay Enhancer Board, Commodore Amiga.

Présentation du standart IFF-ILBM

"IFF" correspond à l'abréviation de "Interchange File Format", et **"ILBM"** correspond à "Interleaved Bitmap". IFF est un format de fichier utilisé pour l'échange de données entre applications en offrant une structure standard de fichiers quel que soit leur contenu: code de programme, données graphiques, sonores, textuelles,...

Le format IFF est un format standard de fichiers développé par la firme Electronic Arts qui permet ainsi à différentes applications de s'échanger des données.

Le début d'un fichier IFF est constitué par le mot "FORM", ce qui signifie le début d'un secteur de données correspondant à la forme déterminée (texte, image, son,...). Ce mot est ensuite suivi par 32 bits indiquant la longueur de cette forme en octets. Après cela, on retrouve 4 caractères indiquant le type de la forme (WORD: texte, ILBM: graphique, SMUS: musique,...).

Quant aux caractéristiques ou composantes des fichiers au format ILBM, elles sont le BitmapHeader (BMHD), la ColorMap (CMAP), les Bits-planes (BODY), le mode ViewPort de l'Amiga (CAMG), les données Graphicraft Colorcycle (CCRT).

* *Le BitmapHeader*

Il détermine la largeur et la hauteur du graphique, le nombre de plans de bits, l'utilisation ou non d'un algorithme de compression, la couleur transparente, etc...

**La ColorMap*

Elle détermine la quantité de rouge, de vert et de bleu pour chaque couleur de la palette.

**Les Bits-Planes*

Ils décrivent ensemble le contenu de chaque plan de bits constituant, dans leur ensemble, le graphique.

**Le mode ViewPort de l'Amiga*

Il détermine dans quel mode reconnu par l'Amiga le graphique doit être affiché (par exemple, Hi-Res, Lo-Res, Lace, etc).

**Les données Graphicraft Colorcycle*

Ces données fournissent des informations concernant la manière dont doit se dérouler le "cyclage" des couleurs si on désire utiliser cet effet.

VIII.6. CONCLUSIONS

Dans ce chapitre, nous avons passé en revue, de façon superficielle, quelques-unes des notions concernant le graphisme par ordinateur qui sont primordiales si l'on veut comprendre comment a été réalisé l'apllication de "Logiciel BD", du point de vue graphique. Cet aspect de la réalisation du logiciel a demandé un travail important de recherche d'informations dans le domaine du graphisme par ordinateur et a nécessité la maîtrise de l'Amiga à des niveaux fort proche du "hardware". En effet, le langage que nous avons utilisé pour programmer le logiciel (Langage C) offre la possibilité de travailler sur des primitives graphiques de bas niveau sans pour autant offrir une interface pour des primitives de haut niveau, ce qui n'a pas facilité le travail d'implémentation.

Conclusion

La conception d'un logiciel, permettant à des personnes mentalement déficientes d'exprimer leur créativité en réalisant des pages de bandes dessinées, est une tâche très complexe. Elle a d'ailleurs nécessité la récolte de connaissances dans des disciplines fort diverses.

Etant donnée la spécificité de la population utilisatrice, il a tout d'abord été nécessaire de s'intéresser à l'aspect psychologique du logiciel. Nous avons ainsi dû déterminer la partie de cette population et ses caractéristiques, pour laquelle notre logiciel devait être conçu; et ce, afin de prendre conscience des capacités et des exigences de celle-ci. C'est ainsi qu'il a été décidé, dès le début, que le logiciel s'adresserait uniquement aux personnes modérément ou faiblement handicapées mentales. Afin de bien adapter le logiciel à la population cible, nous avons non seulement porté beaucoup d'attention aux fonctionnalités du logiciel mais aussi, et surtout, à l'adéquation de l'interface homme/machine à cette population. Nous avons ainsi fait appel à des connaissances concernant le domaine de conception d'interfaces utilisateurs. Cependant, notre logiciel ne peut en aucun cas remplacer le manque de créativité graphique chez une personne qu'elle soit dite normale ou mentalement handicapée. Il doit tout au plus être apparenté à un bras de levier aux forces démultiplicatrices.

Une autre matière, nouvelle pour nous, fut celle ayant trait aux manipulations d'images graphiques par ordinateur.

Finalement, la réalisation de "Logiciel BD" nous a permis de découvrir une méthode de conception de logiciels que nous ne connaissions pas, à

savoir la conception orientée objet. Cette conception se base sur une théorie mettant en oeuvre des concepts et mécanismes qui nous étaient relativement inconnus. Avec la partie relative à la fixation des spécifications de "Logiciel BD", cette conception fut une des plus longues étapes de notre travail: elle nécessita également plusieurs mois. De plus, vu le nombre important d'étapes que propose la méthode orientée objet suivie, nous avons décidé de ne pas la respecter entièrement; au risque sinon de s'engager dans un travail de très longue haleine. Toutefois, ces théorie et méthode orientées objet, quoique plus difficiles que l'approche "traditionnelle", disposent de certains avantages non négligeables dont le plus important est celui de réutilisabilité.

Comme on le voit, les domaines que nous avons abordés ne nous étaient, pour la plupart, pas familliers. C'est la raison pour laquelle la fixation précise des objectifs du logiciel ainsi que des limites de ce qu'il fallait réaliser s'est toujours faite en tenant compte de l'expérience que nous avons eue avec les personnes mentalement déficientes, mais aussi des capacités techniques des machines avec lesquelles nous allions travailler. C'est selon cette démarche que nous avons voulu réaliser un cahier des charges qui soit le plus complet possible et qui propose un projet très "ouvert".

Etant donnée l'ampleur du projet "Logiciel BD", tel que le prévoyait le cahier des charges, nous n'avons pu qu'implémenter un seul sous-système de celui-ci. Il s'agit précisément de l'application "Préparation Page BD" de la structuration des traitements. Celle-ci concerne le travail de création proprement-dit d'une page de bandes dessinées par un utilisateur mentalement déficient accompagné d'un animateur. De plus, comme la plupart des nombreux "outils" permettant de manipuler les graphiques d'une page de bandes dessinées qui étaient proposés, dans ce même cahier des charges, nécessitaient une implémentation complexe, nous avons décidé de programmer ceux qui étaient les plus indispensables, dans un premier temps, au bon fonctionnement du logiciel.

Les premières extensions à l'implémentation de "Logiciel BD" qui viennent à l'esprit sont celles qui concernent les éléments du cahier des charges qui n'ont pas encore été développés. Nous pensons, en effet, que ce cahier des charges est suffisament complet pour concerner un projet de grande envergure. Cependant, d'autres idées d'extension pour le logiciel

nous paraissent être pertinentes. Ainsi, l'implémentation de la couleur pourrait être intéressante, mais nécessiterait alors l'utilisation d'un autre ordinateur que l'Amiga 2000 afin que l'utilisateur ait à sa disposition un grand choix de couleurs. Le logiciel pourrait aussi être adapté afin de permettre la prise en charge de divers périphériques d'ordinateurs tels que, par exemples, des tablettes graphiques, des crayons optiques,...

Bien que nous n'ayons malheureusement pas eu le temps d'évaluer l'utilisation du logiciel auprès de personnes mentalement déficientes, il sera possible de l'entreprendre ultérieurement. Comme un animateur assistera toujours l'utilisateur, nous pensons que l'évaluation peut être réalisée par ce premier qui observera la manière dont sont maîtrisés les outils ainsi que les concepts liés à chacun de ceux-ci. Ainsi, la création d'un logiciel conservant les traces des actions des utilisateurs ne serait pas indispensable dans un premier temps. C'est ainsi que les outils dont l'utilisation serait jugée trop complexe pourraient être modifiés.

Finalement, l'évaluation du prototypage de l'interface utilisateur, ayant rencontré un franc succès auprès de la population concernée, nous estimons que le logiciel devrait être bien accueilli.

Bibliographie

[AVLAM,90] VAN LAMSWEERDE A.,*Méthodologie de Développement de Logiciels,* Notes de cours, Deuxième licence, FNDP Institut d'Informatique, Namur, 1989-90.

[BARTH,88] BARTHET M.-F., *Logiciels interactifs et ergonomie modèles et méthodes de conception,* Dunod (informatique), Edition Bordas, Paris, 1988.

[BODAR,90] BODART F., *Cours introductif aux interfaces hommemachine : notes de cours provisoires,* FNDP Namur, 1989-1990.

[BODPI,89] BODART F., PIGNEUR Y., *Conception assistée des systèmes d'information : Méthode - Modèles - Outils,* 2nd édition, MIPS, Masson, 1989.

[BODPR,89] BODART F., PROVOT I., *Eléments de modélisation et d'architecture des Interfaces homme-machine dans les systèmes d'information : une approche pragmatique,*troisième cycle roman d'informatique, Lausanne, Septembre1989.

[BOLAM,81] BOULANGE L., LAMBERT J.L., *Les Autres - Expressions artistiques chez les handicapés mentaux,* Pierre Mardaga éditeur, 1981.

[BRABA,89] DE BRABANDERE L.,*Le Latéroscope, Systèmes et Créativité,* La Renaissance du Livre, 1989.

[COUTA,88] COUTAZ J., *Interface Homme-Ordinateur: Conception et réalisation,* Thèse de doctorat présentée à l'Université Joseph Fourier de Grenoble, décembre 1988.

[COX,86] BRAD J. COX, *Object Oriented Programming, An Evolutionary Approach,* Addison-Wesley Publishing Company, 1986.

[FRAIT,90] FRAITURE M., MACHGEELS C., *Le cahier des charges d'un logiciel pour les personnes handicapées,* Hantépsycom, Kerpape, Juillet 1990.

[FRESN,72] FRESNAULT, DERUELLE, La B.D., *Essai d'analyse sémiotique,* Hachette, 1972.

[GROSS,83] GROSSMAN, *Classification in Mental Retardation,* American Association on Mental Deficiency,1983.

[HARTS,89] HARTSON H., HIX D., *Human-computer interface development: concepts and systems for its management,*ACM computing surveys, volume 21, No 1, March 1989.

[HEINT,88] HEINTZ N., *Créativité et handicap mental,*Carrefour 88, Actes du Congrès, Octobre 1988.

[HALBE,87] HALBERT C.H., O'BRIEN P.D., *Usin Types And Inheritance In Object-Oriented Programming,*IEEE Software, September 1987.

[INFPC,91] PIGOT T., *Programmation Orientée Objet,* INFO PC, Octobre 1991.

[IONES,87] IONESCU S., *L'intervention en déficience mentale, Problèmes généraux et méthodes médicales et psychologiques,*Volume 1, Edition Mardaga, Bruxelles, 1987.

[KAPTA,78] KAPTAN H., *Le Processus de Création Picturale des Personnes Handicapées Mentales,*Paris, SNADOC, 1978.

[KHOSH,90] KHOSHAFIAN S., ABNOUS R., *Object Orientation, Concepts, Languages, Databases, User Interface,*Wiley Professional Computing, 1990.

[LAMBE,78] LAMBERTJ-L., *Introduction à l'arriération mentale,* Pierre Mardaga (Editeur), Collection Psychologie et sciences humaines, Bruxelles, 1978.

[LAMBE,86] LAMBERTJ-L., *Handicap mental et société: un défi pour l'éducation,* Delval (Editions), Cousset (Fribourg), Suisse, 1986.

[LECHA,91] LECHARLIER B., *Programmation Orientée Objet (en Turbo Pascal),* Cours de 2nd Candidature en Sciences Economiques option Informatique, F.N.D.P. Namur, 1991.

[LEPOU,90] LEPOUTRE T., ROQUET J.M., *La personne handicapée mentale et la connaissance du corps humain: un logiciel d'apprentissage,*Mémoire, F.N.D.P. Institut d'informatique, 1989-90.

[MCSYS,90] REICH G.-P., *Les cahiers du développeur,*Micro Systèmes n°114, Décembre 1990.

[MEMBG,90] BROUSMICHE L., GILLET X., *Logiciel de simulation d'un jeu de marionnettes pour enfants infirmes moteurs cérébraux,* Mémoire, F.N.D.P. 1989-1990.

[MERWI,91] MERCIER M., WITDOUCK O., *La créativité sur ordinateur pour la personne handicapée mentale,* Journal de Réfexion sur l'Informatique n°20, Juin 1991.

[MEYER,88] MEYER B., *Object-Oriented Software Construction,* C.A.R.HOARE SERIES EDITOR.

[MEYER,87] MEYER B., *Reusability : The Case For Object-Oriented Design,* IEEE Software, March 1987.

[MICSS,90] HOUBART G., *Interfaces Homme-Machine: La Programmation recule au profit de l'interactivité,* Micro Systèmes n°105, Février 1990.

[NEWMA,79] WILLIAM M., NEWMAN, ROBERT F. SPROULL, *Principles of Interactive Computer Graphics,* Second Edition, McGraw-Hill Book Company, 1979.

[NORMA,86] NORMAN D., DRAPER S., *User centered system design : New perspectives on human-computer interaction,* Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1986.

[OSTER,76] OSTERRIETH P.A., *Le dessin chez l'enfant.,* In Gratiot-Alphandéry, H. et Zazzo, R. (Eds.), Paris:Presses Universitaires de France, 1976.

[PETOU,90] PETOUD I., *Génération automatique de l'interface homme-machine d'une application de gestion hautement interactive : Thèse,*Université de Lausaunne, Ecole des hautes études commerciales, 1990.

[SCAPI,86] SCAPIN D.L., *Guide ergonomique de conception des interfaces homme-machine,* rapports techniques, N°77, Institut National de recherche en informatique et en automatique, Octobre 1986.

[SCHIL,87] SCHILDT H., *C, The Complete Reference,* Osborne McGraw-Hill, 1987.

[SCHNE,87] SCHNEIDERMAN B., *Designing the user interface strategies for effective human-computer interaction,* Addison-Wesley Publishing Company, 1987.

[STROU,88] STROUSTRUP B., *What Is Object-Oriented Programming,* IEEE Software, May 1988.

[THEVO,75] THEVOZ M., *L'art brut.,* Genève: Editions d'Art Albert Skira, 1975

[UNIV,90] *Encyclopédie Universalis,* Editeur Paris,1990, Tome 1

[VIGLI,83] VIGLIARON A., *Le graphisme chez des enfants handicapés mentaux - essai d'activité,* Institut d'Etudes Sociales, 1983.

[WEIZE,81] WEIZENBAUM J., *Puissance de l'ordinateur et raison de l'homme: du jugement au calcul,* Informatique (Edition), 1981.

[WIRFS,90] WIRFS-BROCK R., WILKERSON B., WIENER L., *Designing Object Oriented Software,* Prentice Hall, Englewood Cliffs, New Jersey, 1990

[ZIGLE,84] ZIGLER E., BALLA D., HODAPP R., *On the definition and classification of mental retardation,* American Journal of Mental Deficiency, 1984, 215-230.

Réalisation, impression et reliure : SPRL Centre de Reprographie - Tél. : 081/22.87.72

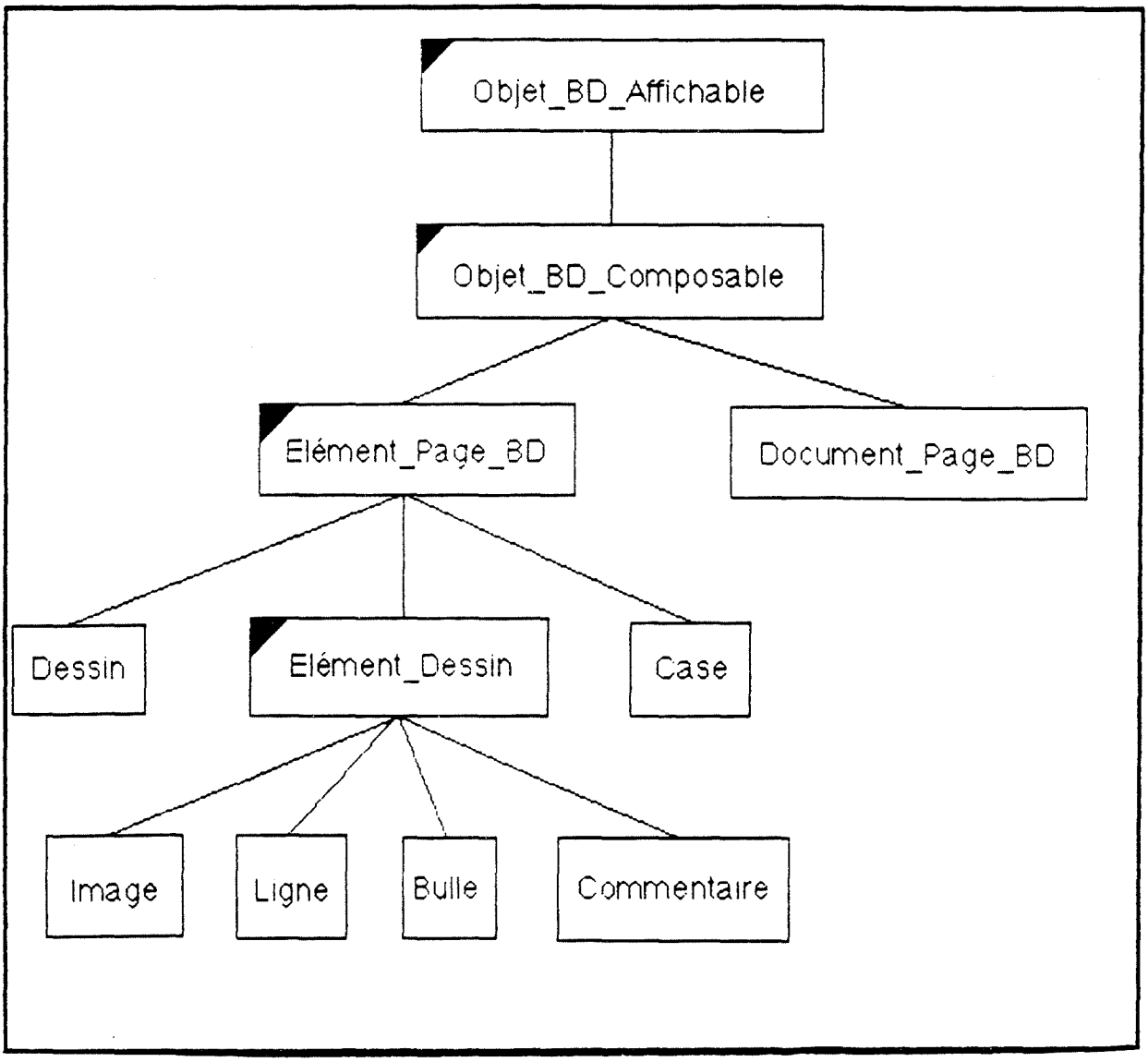
Table des matières

Découpe orientée objet.

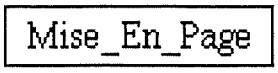
SOUS-SYSTEME DOCUMENT PAGE BD

Ce sous-système encapsule les détails de représentation d'un document page BD.

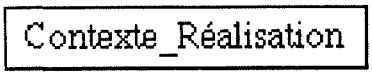
Graphe hiérarchique de Objet BD Affichable.



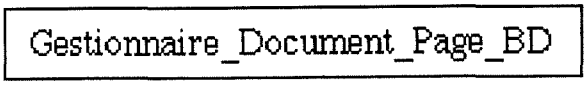
Graphe hiérarchique de Mise En Page.



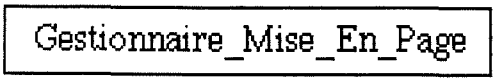
Graphe hiérarchique de Contexte Réaliation.



Graphe hiérarchique de Gestionnaire Document Page BD.



Graphe hiérarchique de Gestionnaire Mise En Page.



Classe: **Objet_BD_Affichable.**

Super-Classe(s): Aucune.

Sous-Classe(s): Objet_BD_Composable.

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Un OBJET BD AFFICHABLE est un objet propre au domaine de la réalisation d'une page de bandes dessinées qui peut s'afficher sur n'importe quel support d'affichage, tel qu'une imprimante, un écran ou une fenêtre de cet écran (voir le graphe hiérarchique d'ELEMENT_INTERFACE).

Contrats :

S'afficher.
S'afficher .

Afficher(Support_Affichage, Position)

utilise: Support_Affichage.

Cette méthode affiche l'OBJET BD AFFICHABLE à l'endroit déterminé par Position sur le support d'affichage. Si l'entièreté de l'objet à afficher ne peut l'être sur le Support d'Affichage, l'objet sera "coupé" afin que seule la partie pouvant être affichée soit visible.

Responsabilités Privées: /

Classe: **Objet_BD_Composable.**

Super-Classe(s): Objet_BD_Affichable.

Sous-Classe(s): Elément_PageBD, Document_PageBD.

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Un OBJET BD COMPOSABLE est un objet propre au domaine de la réalisation d'une page de bandes dessinées qui peut se composer sur un support d'affichage. La composition est une opération qui ajuste la taille rectangulaire maximum (en points du support d'affichage) que l'objet occupe dans une zone rectangulaire déterminée dont la taille est aussi donnée en points du support d'affichage. Au moment de sa composition, un OBJET BD COMPOSABLE détermine cette zone de composition sans réellement s'afficher lui-même en fournissant le plus grand rectangle conservant les proportions de l'objet et contenu entièrement dans la zone rectangulaire pé-citée.

Contrats:

S'afficher.
Ce contrat est hérité de Objet_BD_Affichable.

Se composer.
Se composer.

> **Compose(Support_Affichage, Rectangle) renvoie un Rectangle.**
>
> Utilise: Support_Affichage.
>
> Cette méthode fournit la taille rectangulaire maximum (en points du support d'affichage) que l'objet occupe dans la zone rectangulaire donnée (sa taille est aussi donnée en points du support d'affichage). Elle fournit, en fait, le plus grand rectangle conservant les proportions de l'objet et contenu entièrement dans la zone rectangulaire pé-citée.

Responsabilités Privées: /

Classe: **Document_Page_BD.**

Super-Classe(s): Objet_BD_Composable.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Cette classe représente une page de bandes dessinées. Elle représente la structure regroupant différents éléments qui composent une page de bandes dessinées. Un DOCUMENT PAGE BD contient une liste de DESSINS. Chaque dessin qui la compose a une position unique sur cette page. Cette position est fixée par la MISE EN PAGE associée au DOCUMENT PAGE BD. Le nombre de DESSINS et leur position dans la page blanche reste fixe et sont aussi imposés par la MISE EN PAGE associée.

Contrats:

Se composer.
Ce contrat est hérité de Objet_BD_Composable.

S'afficher.
Ce contrat est hérité de Objet_BD_Affichable.

S'effacer.
S'effacer.

Effacer(Fenêtre).

Utilise: Fenêtre.

Cette méthode commande l'effacement de la page de bandes dessinées actuellement affichée dans une fenêtre.

Mettre à jour et connaître les informations concernant les DESSINS.
Connaître et manipuler les informations concernant tous les DESSINS.

Get_Dessin(Numéro_Case) renvoie un DESSIN.

Utilise : Case, Dessin.

Cette méthode fournit le DESSIN contenu dans la CASE de la MISE EN PAGE et de numéro donné .

Get_Nombre_Dessins_Non_Vide() renvoie un Entier.

Utilise: Dessin.

Cette méthode fournit le nombre de DESSINS non vides contenus dans le DOCUMENT PAGE BD.

Détruire_Dessins().

Utilise: Dessin, Case, Mise_En_Page.

Cette méthode permet de détruire tous les DESSINS de la page de bandes dessinées.

Connaître les informations concernant la MISE EN PAGE associée.

Get_MEP() renvoie une MISE EN PAGE.

Utilise: Mise_En_Page, Contexte_Réalisation.

Cette méthode fournit la MISE EN PAGE associée au DOCUMENT PAGE BD.

Responsabilités Privées: /

Classe: **Elément_Page_BD.**

Super-Classe(s): Objet_BD_Composable.

Sous-Classe(s): Elément_Dessin, Dessin, Case.

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Cette classe représente un élément constitutif d'un DOCUMENT PAGE BD.

Contrats:

Se composer.
Ce contrat est hérité de Objet_BD_Composable.

Gérer l'affichage de l'objet.
Ce contrat est en partie hérité de Objet_BD_Affichable.

Mise_Evidence(Fenêtre,Position).

Utilise: Fenêtre.

Cette méthode affiche l'élément avec un contour de mise en évidence à l'endroit spécifié par Position.

Déplacer(Fenêtre, Position).

Utilise: Fenêtre.

Cette méthode déplace l'élément affiché dans la FENETRE donnée à la POSITION donnée.

Connaître et manipuler l'état de l'ELEMENT PAGE BD.
Connaître et manipuler l'état de sélection de l'ELEMENT PAGE BD.

Sélectionner().
Cette méthode permet de sélectionner l'élément. Si l'élément est déjà sélectionné, son état ne change pas.

Désélectionner().
Cette méthode permet de désélectionner l'élément. Si l'élément n'était pas sélectionné précédemment, son état ne change pas.

Etat_Sélection() renvoie un Booléen.
Cette méthode fournit l'état de sélection de l'élément.Si l'élément est sélectionné, le booléen = Vrai, il sera égal à Faux sinon.

Responsabilités Privées:

- Connaître le motif de mise en évidence du contour de l'Elément.

Classe: **Dessin.**

Super-Classe(s): Elément_Page_BD.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Un dessin est le contenu graphique d'une case de la page de BD. Il s'agit d'une collection (liste) d'ELEMENTS DESSIN qui sont organisés logiquement en niveaux de plans.

Contrats:

Se composer.
Ce contrat est hérité de Objet_BD_Composable.

S'afficher.
Ce contrat est en partie hérité de Objet_BD_Affichable.

Mise_Evidence(Fenêtre,Position).

Utilise: Fenêtre.

Cette méthode affiche le dessin avec un contour de mise en évidence à l'endroit spécifié par Position.

Déplacer(Fenêtre, Position).

Utilise: Fenêtre.

Cette méthode déplace le dessin affiché dans la FENETRE donnée à la POSITION donnée.

Imprimer_Dessin() renvoie un Booléen.

Utilise: Element_Dessin.

Cette méthode envoie le contenu graphique du DESSIN à l'imprimante. Elle renvoie un Booléen égal à "Faux" si une erreur s'est produite durant l'impression.

Manipuler la liste des ELEMENTS du DESSIN.
Savoir quels éléments sont contenus dans le Dessin.

Nouvel_Elément(Elément_Dessin).

Utilise: Element_Dessin.

Cette méthode permet d'ajouter un nouvel ELEMENT DESSIN dans la liste.

Element_A(Position).

Utilise: Element_Dessin.

Cette méthode renvoie le premier élément dans la liste des Eléments_Dessin dont le contour rectangulaire contient le point spécifié par Position. Si aucun élément ne contient ce point, un objet nul est renvoyé.

Nombre_Eléments() renvoie un Entier.

Cette méthode fournit le nombre d'éléments dessin de la liste.

Premier_Eléments() renvoie une IMAGE.

Utilise: Elément_Dessin.

Cette méthode fournit la référence du premier ELEMENT DESSIN de la liste.

Détruire_Liste().

Utilise: Elément_Dessin.

Cette méthode détruit toutes les occurences des ELEMENTS DESSIN de la liste.

Détruire_Elément(Numéro_Ordre).

Utilise: Elément_Dessin.

Cette méthode détruit l'ELEMENT DESSIN de la liste référencé par le Numéro d'Ordre donné.

Connaître et manipuler l'état du DESSIN.
Ce contrat est hérité de Elément_Page_BD.

Connaître des informations concernant le DESSIN.
Fournir la couleur de fond du dessin.

Couleur_Fond() renvoie un entier positif.

Cette méthode fournit la couleur de fond du dessin. Il s'agit, en fait, d'une valeur entière représentant une couleur de la palette de couleurs utilisée.

Responsabilités Privées:

- Connaître la case à laquelle appartient le dessin.

Classe: **Elément_Dessin.**

Super-Classe(s): Elément_Page_BD.

Sous-Classe(s): Image, Ligne, Bulle, Commentaire.

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Cette classe représente un ELEMENT constitutif d'un DESSIN. Un ELEMENT d'un DESSIN est un ELEMENT de la liste gérée par ce DESSIN.

Contrats:

Se composer.
Ce contrat est hérité de Objet_BD_Composable.

S'afficher.
Ce contrat est en partie hérité de Objet_BD_Affichable.

Mise_Evidence(Fenêtre,Position).

Utilise: Fenêtre.

Cette méthode affiche l'élément dessin avec un contour de mise en évidence à l'endroit spécifié par Position.

Déplacer(Fenêtre, Position).

Utilise: Fenêtre.

Cette méthode déplace l'élément dessin affiché dans la FENETRE donnée à la POSITION donnée.

Connaître et manipuler les informations concernant l'ELEMENT DESSIN.
Connaître et manipuler l'état de sélection de l'ELEMENT DESSIN.
Cette responsabilité est héritée de Elément_Page_BD.

Connaître les informations concernant l'ELEMENT DESSIN.

Dessin_Contenant() renvoie un DESSIN.

Utilise: Dessin.

Cette méthode fournit le DESSIN auquel est associé l'ELEMENT DESSIN.

Responsabilités Privées: /

Classe: **Case.**

Super-Classe(s): Elément_Page_BD

Sous-Classe(s): Aucune

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Cette classe représente une CASE d'une MISE EN PAGE. Il s'agit du contour rectangulaire du DESSIN correspondant dans la PAGE de BD.

Contrats:

Se composer.
Ce contrat est hérité de Objet_BD_Composable.

S'afficher.
Ce contrat est en partie hérité de Objet_BD_Affichable.

Mise_Evidence(Fenêtre,Position).

Utilise: Fenêtre.

Cette méthode affiche la case avec un contour de mise en évidence à l'endroit spécifié par Position.

Déplacer(Fenêtre, Position).

Utilise: Fenêtre.

Cette méthode déplace la case affichée dans la FENETRE donnée à la POSITION donnée.

Connaître et manipuler les informations concernant la CASE.
Connaître et manipuler l'état de sélection de la CASE.
Cette responsabilité est héritée de Elément_Page_BD.

Fournir des informations concernant la CASE.

Numéro_Ordre() renvoie un Entier.

Cette méthode fournit le numéro d'ordre de la CASE dans la MISE EN PAGE du DOCUMENT PAGE BD manipulé.

Dimensions() renvoie un Rectangle.

Cette méthode fournit une structure de type RECTANGLE contenant les dimensions de la CASE en mm. Un structure RECTANGLE est composée respectivement des entiers Largeur et Hauteur.

Responsabilités Privées:

- Connaître le motif du contour de la CASE.
- Connaître la couleur du contour de la CASE.

Classe: **Image.**

Super-Classe(s): Elément_Dessin.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Cette classe représente une IMAGE utilisée lors de la création d'un DESSIN de la PAGE de BD en cours de réalisation. Cette IMAGE comporte une représentation graphique sous forme de bitmap. Elle possède un contour rectangulaire qui détermine les limites maximales de cette IMAGE.

Contrats:

Se composer.
Ce contrat est hérité de Objet_BD_Composable.

S'afficher.
Ce contrat est en partie hérité de Objet_BD_Affichable.

Mise_Evidence(Fenêtre,Position).

Utilise: Fenêtre.

Cette méthode affiche l'image avec un contour de mise en évidence à l'endroit spécifié par Position.

Déplacer(Fenêtre, Position).

Utilise: Fenêtre.

Cette méthode déplace l'image affichée dans la FENETRE donnée à la nouvelle POSITION donnée.

Connaître et manipuler les informations concernant l'IMAGE.
Connaître et manipuler l'état de sélection de l'IMAGE.
Cette responsabilité est héritée de Elément_Page_BD.

Manipuler la représentation graphique de l'IMAGE.
Fournir une nouvelle représentation graphique de l'image.

Agrandir(Delta).

Cette méthode augmente la taille de la représentation graphique de l'IMAGE en fonction de Delta (Delta est une valeur réelle positive indiquant le pourcentage d'agrandissement de la taille de l'IMAGE).

Réduire(Delta).

Cette méthode réduit la taille de la représentation graphique de l'IMAGE en fonction de Delta (Delta est une valeur réelle positive indiquant le pourcentage de réduction de la taille de l'IMAGE).

Rotation(Delta, Sens).

Cette méthode effectue une rotation de la représentation graphique de l'IMAGE en fonction du Delta (Delta est une valeur comprise entre 0 et 360 indiquant l'angle de rotation) et du sens donnés (gauche ou droite).

Découper(Contour) renvoie une IMAGE.

Cette méthode crée un nouvelle IMAGE. Cette nouvelle image aura la représentation graphique correspondant à la partie de l'ancienne représentation graphique délimitée par le Contour donné (le contour est un ensemble de coordonnées de points contigus).

Dupliquer() renvoie une IMAGE.

Cette méthode crée un nouvelle instance d'IMAGE ayant les mêmes attributs et la même représentation graphique que cette IMAGE.

Remplir(Position, Couleur).

Cette méthode donne la couleur donnée au point (pixel) désigné par Position, ainsi qu'à tous les points de même couleur contigus à ce points. Le type Couleur est, en fait, un entier représentant une couleur dans la palette de couleur actuelle.

Responsabilités Privées:

- Connaître son état d'association à un DESSIN.

Classe: **Bulle.**

Super-Classe(s): Elément_Dessin.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Cette classe représente un philactère de bandes dessinées. Le contour du philactère est de forme ovale avec une "queue" en dessous pour déterminer l'origine du dialogue.

Contrats:

Se composer.
Ce contrat est hérité de Objet_BD_Composable.

S'afficher.
Ce contrat est en partie hérité de Objet_BD_Affichable.

Mise_Evidence(Fenêtre,Position).

Utilise: Fenêtre.

Cette méthode affiche la bulle avec un contour de mise en évidence à l'endroit spécifié par Position.

Déplacer(Fenêtre, Position).

Utilise: Fenêtre.

Cette méthode déplace la bulle affichée dans la FENETRE donnée à la nouvelle POSITION donnée.

Connaître et manipuler les informations concernant la BULLE.
Connaître et manipuler l'état de sélection de la BULLE.
Cette responsabilité est héritée de Elément_Page_BD.

Connaître et modifier le contenu de la bulle.

Fournir_Texte() renvoie du Texte.
Cette méthode retourne le texte contenu dans la BULLE.

Modifier_Texte(Texte).
Cette méthode modifie la représentation graphique de la BULLE avec le nouveau Texte donné.

Responsabilités Privées:

- Connaître son texte.
- Connaître la police de caractères.

Classe: **Commentaire.**

Super-Classe(s): Elément_Dessin.

Sous-Classe(s): Aucune

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Cette classe représente un COMMENTAIRE de BD contenant du texte. Le contour du COMMENTAIRE sera de forme rectangulaire.

Contrats:

Se composer.
Ce contrat est hérité de Objet_BD_Composable.

S'afficher.
Ce contrat est en partie hérité de Objet_BD_Affichable.

Mise_Evidence(Fenêtre,Position).

Utilise: Fenêtre.

Cette méthode affiche le commentaire avec un contour de mise en évidence à l'endroit spécifié par Position.

Déplacer(Fenêtre, Position).

Utilise: Fenêtre.

Cette méthode déplace le commentaire affiché dans la FENETRE donnée à la nouvelle POSITION donnée.

Connaître et manipuler les informations concernant le COMMENTAIRE.
Connaître et manipuler l'état de sélection du COMMENTAIRE .
Cette responsabilité est héritée de Elément_Page_BD.

Connaître et modifier le contenu du commentaire.

Fournir_Texte() renvoie du Texte.
Cette méthode retourne le texte contenu dans le COMMENTAIRE.

Modifier_Texte(Texte).
Cette méthode modifie la représentation graphique du COMMENTAIRE avec le nouveau Texte donné.

Responsabilités Privées:

- Connaître son texte.
- Connaître la police de caractères.

Classe: **Ligne.**

Super-Classe(s): Elément_Dessin.

Sous-Classe(s): Aucune

Graphe Hiérarchique: Objet_BD_Affichable.

Description: Cette classe représente un objet graphique ayant les caractéristiques géométriques d'une LIGNE droite.

Contrats:

Se composer.
Ce contrat est hérité de Objet_BD_Composable.

S'afficher.
Ce contrat est en partie hérité de Objet_BD_Affichable.

Mise_Evidence(Fenêtre,Position).

Utilise: Fenêtre.

Cette méthode affiche la ligne avec un contour de mise en évidence à l'endroit spécifié par Position.

Déplacer(Fenêtre, Position).

Utilise: Fenêtre.

Cette méthode déplace la ligne affichée dans la FENETRE donnée à la nouvelle POSITION donnée.

Connaître et manipuler les informations concernant la LIGNE.
Connaître et manipuler l'état de sélection de la LIGNE .
Cette responsabilité est héritée de Elément_Page_BD.

Manipuler le motif de la LIGNE.

Changer_Motif(Motif).

Utilise: Contexte_Réalisation.

Cette méthode change le motif de la LIGNE avec celui fourni comme paramètre. Motif est, en fait, un entier correspondant à un motif appartenant à la liste des motifs disponibles se trouvant dans le contexte de réalisation.

Responsabilités Privées:

- Connaître motif actuel de dessin de la LIGNE.
- Connaître la longueur de la LIGNE en mm.

Classe: **Mise_En_Page.**

Super-Classe(s): Aucune.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Mise_En_Page.

Description: Cette classe représente une MISE EN PAGE qui pourra être utilisée pour une PAGE de BD. Il s'agit de l'assemblage de CASES sur une page blanche de dimensions données. Elle gère la liste des CASES qu'elle contient.

Contrats:

Mettre à jour et connaître les informations concernant les CASES.
Connaître et manipuler les informations concernant toutes les CASES.

Get_Case(Numéro) renvoie une CASE.

Utilise: Case.

Cette méthode fournit la CASE ayant le Numéro de CASE donné.

Get_Nombre_Cases() renvoie un Entier.

Cette méthode fournit le nombre de CASES contenues dans la MISE EN PAGE.

Détruire_Cases().

Utilise: Case.

Cette méthode permet de détruire toutes les CASES de la liste.

Get_Position(Case) renvoie Position.

Utilise: Case.

Cette méthode fournit la Position d'une CASE donnée dans la MISE EN PAGE.

Get_Dimensions(Case) renvoie Dimensions.

Utilise: Case.

Cette méthode renvoie le Dimensions de la CASE donnée.

Ajout_Case(Case, Position) renvoie un Booléen.

Utilise: Case.

Cette méthode permet de créer un nouvelle instance de CASE et de l'ajouter dans la MISE EN PAGE à la Position indiquée par le paramètre Position. Le Booléen renvoyé est égal à Faux si la nouvelle CASE chevauche une ou plusieurs autres CASES de la MISE EN PAGE, dans ce cas aucune nouvelle case n'est créée. Il est égal à Vrai sinon.

Supprimer_Case(Case).

Utilise: Case

Cette méthode permet de supprimer de la MISE EN PAGE une CASE déterminée.

Get_Dimensions_Page() renvoie des Dimensions.

Cette méthode fournit les dimensions de la PAGE sur laquelle se trouve la MISE EN PAGE.

Responsabilités Privées:

- Connaît les dimensions de la page blanche.
- Connaît le nombre d'objets de la liste.
- Connaît la position dans la page blanche de chaque case de la liste.

Classe: **Contexte_Réalisation.**

Super-Classe(s): Aucune.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Contexte_Réalisation.

Description: Un contexte de réalisation représente l'état actuel du système lors de la réalisation d'une PAGE BD. La réalisation se fait dans un contexte de styles de texte (police de caractères) et une MISE EN PAGE en vigueur à un moment donné. Il est utilisé en tant que mécanisme pour la communication entre les objets qui peuvent se composer et leurs clients. Les contextes de réalisation sont conçus pour représenter tous les facteurs qui déterminent les contraintes appliquées à n'importe quel objet du sous-système "Document Page BD". Chacun de ces objets est responsable de requérir cette contrainte au contexte de réalisation en ce qui concerne certains attributs qui déterminent comment il doit être affiché.
De plus, ce contexte de réalisation doit connaître les informations concernant l'UTILISATEUR actuel afin que les accès se fassent sans erreur dans sa ZONE DE TRAVAIL. Le CONTEXTE DE REALISATION sera créé en début de chaque session de travail d'un UTILISATEUR.

Contrats:

Connaître et Mettre à jour les paramètres de réalisation.
Connaître et Mettre à jour le style actuel de la Mise En Page.

Get_MEP() renvoie une Mise_En_Page.

Utilise: Mise_En_Page.

Cette méthode renvoie la MISE EN PAGE de la PAGE BD actuelle.

Set_MEP(Mise_En_Page).

Utilise: Mise_En_Page, Page_BD.

Cette méthode crée une instance de PAGE BD et y associe la MISE EN PAGE choisie.

Connaître et Mettre à jour le style actuel du Texte.

Set_Style_Texte(Style_Texte).

Cette méthode fixe le Style du Texte (police de caractères) dans l'instance du CONTEXTE DE REALISATION.

Get_Style_Texte() renvoie Style_Texte

Cette méthode fournit le Style de Texte (police de caractères) de l'instance du CONTEXTE DE REALISATION.

Connaître et Mettre à jour le DESSIN actuel.

Get_Dessin_Actuel() renvoie un DESSIN.

Utilise: Dessin.

Cette méthode fournit la référence du DESSIN actuel.

Set_Dessin_Actuel(Dessin).

Utilise:Dessin.

Cette méthode permet de sélectionner un nouveau DESSIN actuel.

Connaître et Mettre à jour la CASE actuelle.

Get_Case_Actuelle() renvoie une CASE.

Utilise: Case.

Cette méthode fournit la référence de la CASE actuelle.

Set_Case_Actuelle(Case).

Utilise: Case.

Cette méthode permet de sélectionner une nouvelle CASE actuelle.

Connaître les informations concernant l'UTILISATEUR actuel.

Get_Utilisateur_Actuel() renvoie un UTILISATEUR.

Utilise: Utilisateur.

Cette méthode fournit la référence de l'UTILISATEUR qui travaille actuellement avec le "Logiciel BD".

Set_Utilisateur_Actuel(UTILISATEUR).

Utilise: Utilisateur.

Cette méthode fixe l'UTILISATEUR qui travaille avec le "Logiciel BD".

Connaître les informations concernant le DOCUMENT PAGE BD actuel.

Get_Document_Page_BD_Actuel() renvoie un DOCUMENT PAGE BD.

Utilise: Document_Page_BD.

Cette méthode fournit la référence du Document Page BD sur lequel on travaille actuellement.

Sauver et restaurer le contexte actuel.

Save_Contexte()

Cette méthode sauve en mémoire secondaire l'état actuel de l'instance du CONTEXTE DE REALISATION pour une utilisation future.

Restaure_Contexte()

Cette méthode restaure le dernier état sauvé du CONTEXTE DE REALISATION.

Responsabilités Privées: /

Classe: **Gestionnaire_Document_Page_BD.**

Super-Classe(s): Aucune.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Gestionnaire_Document_Page_BD.

Description: Un gestionnaire de Document Page BD met à jour la liste des DOCUMENTS PAGE BD se trouvant dans la Zone de Travail de l'UTILISATEUR actuel.

Contrats:

Accéder à un Document Page BD sauvé dans la zone de travail de l'utilisateur.

Charge_Document_Page_BD(Nom Fichier) renvoie un DOCUMENT PAGE BD.

Utilise: Utilisateur, Contexte_Réalisation, Document_Page_BD.

Cette méthode crée un nouveau DOCUMENT PAGE BD à partir des informations contenues dans le fichier de nom donné qui se trouve la Zone de Travail de l'UTILISATEUR Actuel.

Ajouter un nouveau Document Page BD dans la zone de travail de l'utilisateur.

Sauver_Document_Page_BD(Nom Fichier) renvoie un Booléen.

Utilise: Utilisateur, Contexte_Réalisation, Document_Page_BD.

Cette méthode sauve, sous le nom de fichier donné, les informations concernant le DOCUMENT PAGE BD actuel dans la Zone de Travail de l'UTILISATEUR Actuel. Elle renvoie un Booléen égal à faux si l'opération n'a pu être réalisée dans de bonnes conditions.

Responsabilités Privées: /

Classe: **Gestionnaire_Mise_En_Page.**

Super-Classe(s): Aucune.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Gestionnaire_Mise_En_Page.

Description: Un gestionnaire de Mises en Page BD met à jour la liste des MISES EN PAGE BD se trouvant dans la Zone de Travail de l'UTILISATEUR actuel.

Contrats:

Accéder à une Mise en page sauvée dans la zone de travail de l'utilisateur.

Charge_Mise_En_Page(Nom Fichier) renvoie une Mise_En_Page.

Utilise: Utilisateur, Contexte_Réalisation, Mise_En_Page.

Cette méthode crée une nouvelle Mise_En_Page à partir des informations contenues dans le fichier de nom donné qui se trouve la Zone de Travail de l'UTILISATEUR Actuel.

Ajouter une nouvelle Mise en page dans la zone de travail de l'utilisateur.

Sauver_Mise_En_Page(Nom Fichier) renvoie un Booléen.

Utilise: Utilisateur, Contexte_Réalisation, Mise_En_Page.

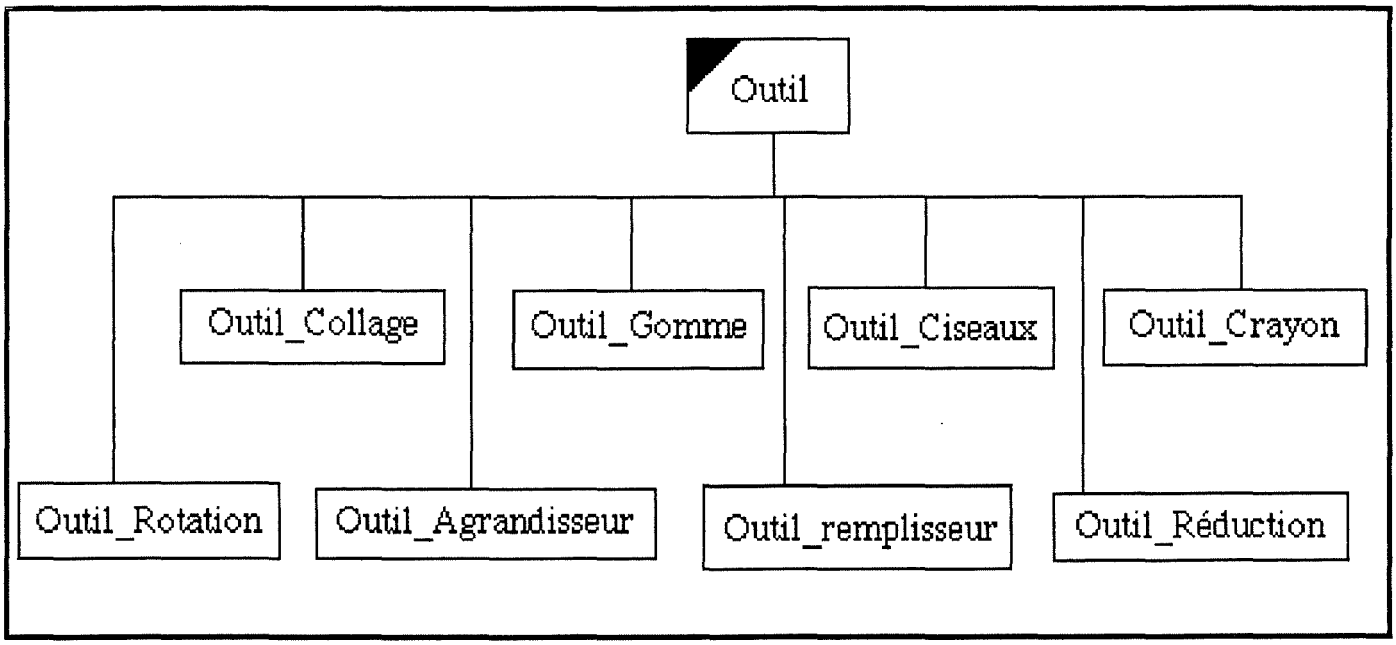
Cette méthode sauve, sous le nom de fichier donné, les informations concernant la Mise en page actuelle dans la Zone de Travail de l'UTILISATEUR Actuel. Elle renvoie un Booléen égal à faux si l'opération n'a pu être réalisée dans de bonnes conditions.

Responsabilités Privées:/

SOUS-SYSTEME EDITEUR GRAPHIQUE

Ce sous-système encapsule l'interprétation des actions de l'utilisateur concernant le travail dans le dessin appartenant à une page de BD.

Graphe hiérarchique de Outil.



Classe: **Outil.**

Super-Classe(s): Aucune.

Sous-Classe(s): Outil_Crayon, Outil_Gomme, Outil_Ciseaux, Outil_Remplisseur, Outil_Collage, Outil_Rotation, Outil_Agrandisseur, Outil_Réducteur.

Graphe Hiérarchique: Outil.

Description: Cette classe représente un OUTIL graphique en général. Un OUTIL permet à un UTILISATEUR de travailler sur le contenu graphique du DESSIN actuel ou de l'IMAGE active.

Contrats:

Lancer l'outil.
Activer l'outil.

Activer().

Cette méthode permet d'activer l'OUTIL. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.

Responsabilités Privées:

- Faire une copie de l'IMAGE active ou du DESSIN actuel en vue d'une récupération future éventuelle lors d'un "undo" (annulation des effets de l'outil).

Classe: **Outil_Crayon.**

Super-Classe(s): Outil

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Outil.

Description: Cette classe représente un outil gérant le dessin point par point dans la représentation graphique (de type bitmap) d'un DESSIN ou d'une IMAGE. Il rendra possible le dessin à main levée au moyen de la souris dans cette représentation graphique selon une taille et une couleur choisie.

Contrats:

Lancer l'outil crayon.
Activer l'outil crayon.

> **Activer_Outil_Crayon(DESSIN).**
>
> Cette méthode permet d'activer l'OUTIL CRAYON pour le travail dans le DESSIN donné. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.
>
> **Activer_Outil_Crayon(IMAGE).**
>
> Cette méthode permet d'activer l'OUTIL CRAYON pour le travail dans l'IMAGE donnée. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.

Responsabilités Privées:
- Sélectionner la couleur de dessin.
- Sélectionner la taille du trait.
- Faire une copie de l'IMAGE active ou du DESSIN actuel en vue d'une récupération future éventuelle lors d'un "undo" (annulation des effets de l'outil).

Classe: **Outil_Gomme.**

Super-Classe(s): Outil_Crayon

Sous-Classe(s): Aucune

Graphe Hiérarchique: Outil.

Description: Cette classe représente un OUTIL permettant d'effacer une partie de la représentation graphique (de type bitmap) d'une IMAGE ou d'un DESSIN. Il agit comme un OUTIL CRAYON qui ne colorie que dans une seule couleur, la couleur de fond du DESSIN.

Contrats:

Lancer l'outil gomme.
Activer l'outil gomme.

> **Activer_Outil_Gomme(DESSIN).**
>
> Cette méthode permet d'activer l'OUTIL GOMME pour le travail dans le DESSIN donné. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.
>
> **Activer_Outil_Gomme(IMAGE).**
>
> Cette méthode permet d'activer l'OUTIL GOMME pour le travail dans l'IMAGE donnée. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.

Responsabilités Privées:
- Sélectionner la couleur de dessin.
- Sélectionner la taille du trait.
- Faire une copie de l'IMAGE active ou du DESSIN actuel en vue d'une récupération future éventuelle lors d'un "undo" (annulation des effets de l'outil).

Classe: **Outil_Ciseaux.**

Super-Classe(s): Outil

Sous-Classe(s): Aucune

Graphe Hiérarchique: Outil

Description: Cette classe représente un OUTIL permettant de découper dans la représentation graphique (de type bitmap) d'une IMAGE ou d'un DESSIN afin d'en extraire une nouvelle IMAGE. Le découpage se fait à main levée en déterminant le contour de la forme de l'IMAGE à découper.

Contrats:

Lancer l'outil ciseaux.
Activer l'outil ciseaux.

> **Activer_Outil_Ciseaux(DESSIN).**
>
> Cette méthode permet d'activer l'OUTIL CISEAUX pour le travail dans le DESSIN donné. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.
>
> **Activer_Outil_Ciseaux(IMAGE).**
>
> Cette méthode permet d'activer l'OUTIL CISEAUX pour le travail dans l'IMAGE donnée. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.

Responsabilités Privées:
- Déterminer le contour à main levée.
- Rendre active l'IMAGE découpée.
- Faire une copie de l'IMAGE active ou du DESSIN actuel en vue d'une récupération future éventuelle lors d'un "undo" (annulation des effets de l'outil).

Classe: **Outil_Remplisseur.**

Super-Classe(s): Outil.

Sous-Classe(s): Aucune

Graphe Hiérarchique: Outil.

Description: Cette classe représente un OUTIL permettant de remplir une zone (ensemble de points contigus et de même couleur) d'une représentation graphique, de type bitmap, d'une IMAGE ou d'un DESSIN, dans une couleur déterminée.

Contrats:

Lancer l'outil remplisseur.
Activer l'outil remplisseur.

Activer_Outil_Remplisseur(DESSIN).

Cette méthode permet d'activer l'OUTIL REMPLISSEUR pour le travail dans le DESSIN donné. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.

Activer_Outil_Remplisseur(IMAGE).

Cette méthode permet d'activer l'OUTIL REMPLISSEUR pour le travail dans l'IMAGE donnée. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.

Responsabilités Privées:
- Remplir la zone déterminée par les coordonnées d'un "point" de celle-ci dans la couleur donnée.
- Faire une copie de l'IMAGE active ou du DESSIN actuel en vue d'une récupération future éventuelle lors d'un "undo" (annulation des effets de l'outil).

Classe: **Outil_Collage.**

Super-Classe(s): Outil.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Outil.

Description: Cette classe représente un OUTIL permettant d'intégrer la représentaion graphique de l'IMAGE active à la représentation graphique du DESSIN actuel à la position choisie par l'utilisateur.

Contrats:

Lancer l'outil collage.
Activer l'outil collage.

> **Activer_Outil_Collage(DESSIN).**
>
> Cette méthode permet d'activer l'OUTIL COLLAGE pour le travail dans le DESSIN donné. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.
>
> **Activer_Outil_Collage(IMAGE).**
>
> Cette méthode permet d'activer l'OUTIL COLLAGE pour le travail dans l'IMAGE donnée. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.

Responsabilités Privées:
- Faire une copie du DESSIN actuel en vue d'une récupération future éventuelle lors d'un "undo" (annulation des effets de l'outil).
- Intégrer la représentation graphique de l'IMAGE active à la représentaion graphique du DESSIN actuel.

Classe: **Outil_Rotation.**

Super-Classe(s): Outil.

Sous-Classe(s): Aucune

Graphe Hiérarchique: Outil.

Description: Cette classe représente un OUTIL permettant d'effectuer une rotation d'angle et de direction choisis à la représentaion graphique de l'IMAGE active.

Contrats:

Lancer l'outil rotation.
Activer l'outil rotation.

> **Activer_Outil_Rotation(IMAGE).**
>
> Cette méthode permet d'activer l'OUTIL ROTATION pour le travail dans l'IMAGE donnée. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.

Responsabilités Privées:

- Faire une copie de l'IMAGE active en vue d'une récupération future éventuelle lors d'un "undo" (annulation des effets de l'outil).
- Demander le sens de la rotation à effectuer.
- Demander l'angle de la rotation à effectuer.
- Provoquer la rotation de la représentation graphique de l'IMAGE active.

Classe: **Outil_Agrandisseur.**

Super-Classe(s): Outil.

Sous-Classe(s): Aucune

Graphe Hiérarchique: Outil.

Description: Cette classe représente un OUTIL permettant d'agrandir la taille de la représentation graphique de l'IMAGE ACTIVE à la demande de l'utilisateur.

Contrats:

Lancer l'outil agrandisseur.
Activer l'outil agrandisseur.

> **Activer_Outil_Agrandisseur(IMAGE).**
>
> Cette méthode permet d'activer l'OUTIL AGRANDISSEUR pour le travail dans l'IMAGE donnée. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.

Responsabilités Privées:

- Faire une copie de l'IMAGE active en vue d'une récupération future éventuelle lors d'un "undo" (annulation des effets de l'outil).
- Demander le taux de variation (DELTA) de l'agrandissement à effectuer.
- Effectuer l'agrandissement.

Classe: **Outil_Réduction.**

Super-Classe(s): Outil.

Sous-Classe(s): Aucune

Graphe Hiérarchique: Outil.

Description: Cette classe représente un OUTIL permettant de réduire la taille de la représentation graphique de l'IMAGE active à la demande de l'utilisateur.

Contrats:

Lancer l'outil réduction.
Activer l'outil réduction.

> **Activer_Outil_Réduction(IMAGE).**
>
> Cette méthode permet d'activer l'OUTIL REDUCTION pour le travail dans l'IMAGE donnée. La méthode permet à l'OUTIL de s'exécuter selon le scénario défini pour celui-ci et en fonction des manipulations de l'utilisateur.

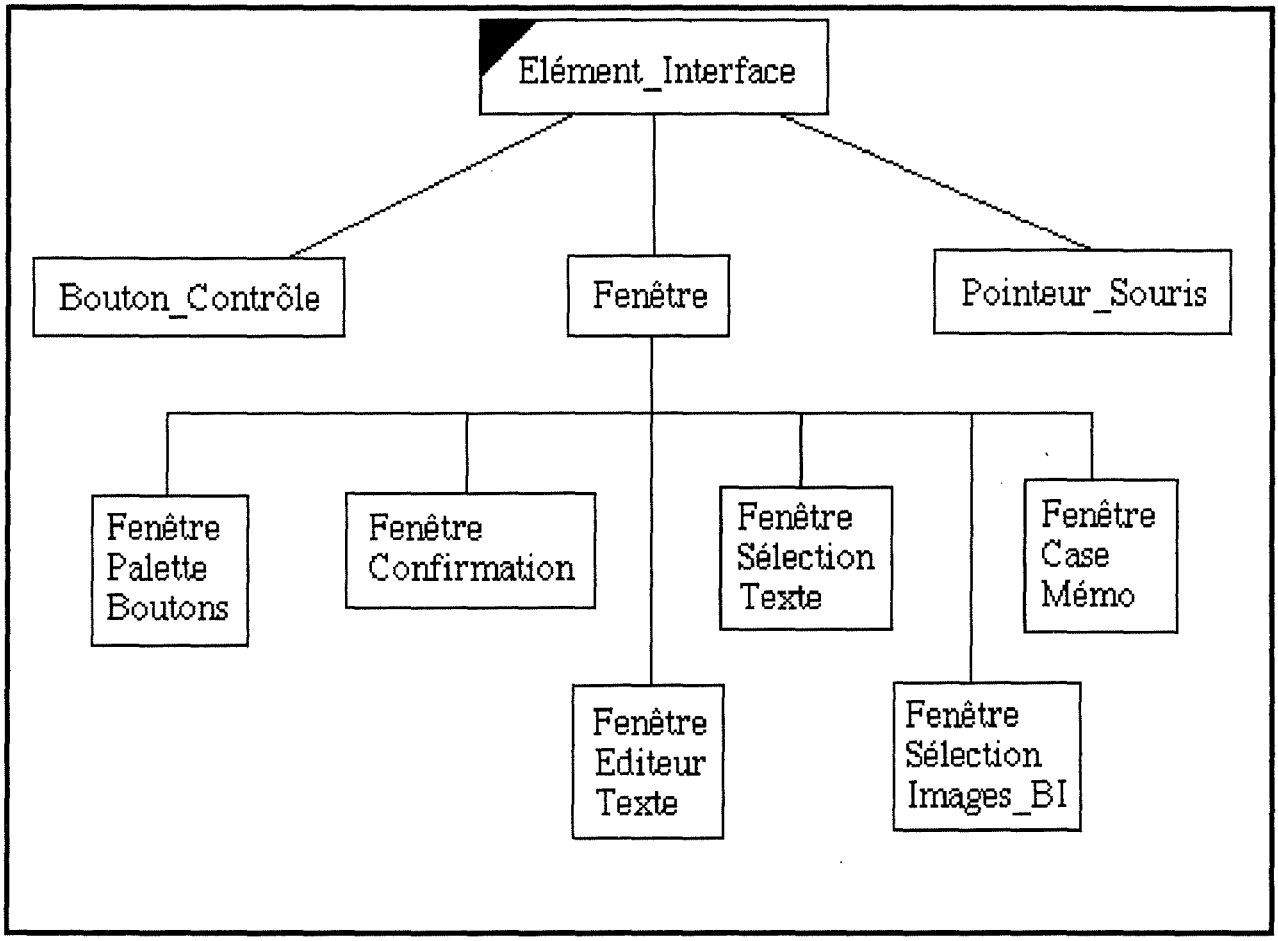
Responsabilités Privées:

- Faire une copie de l'IMAGE active en vue d'une récupération future éventuelle lors d'un "undo" (annulation des effets de l'outil).
- Demander le taux de variation (DELTA) de la réduction à effectuer.
- Effectuer la réduction.

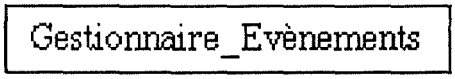
SOUS-SYSTEME INTERFACE UTILISATEUR

Ce sous-système encapsule les éléments de l'interface avec l'utilisateur.

Graphe hiérarchique de Elément Interface.



Graphe hiérarchique de Gestionnaire Evènements.



Classe : **Elément_Interface.**

Super-Classe(s): Aucune.

Sous-Classe(s): Bouton_Contrôle, Fenêtre, Menu, Pointeur_Souris.

Graphe Hiérarchique: Elément_Interface.

Description: Cette classe représente un élément constitutif de l'interface utilisateur graphique. Tous les éléments interface peuvent être manipulés au moyen de la SOURIS.

Contrats:

Gérer l'affichage à l'écran.
Gérer l'affichage de l'ELEMENT INTERFACE à l'écran.

S'afficher(Position).

Cette méthode affiche l'ELEMENT INTERFACE à l'écran à la position donnée.

S'effacer().

Cette méthode efface l'ELEMENT INTERFACE de l'écran.

Connaître les informations concernant l'ELEMENT INTERFACE.
Connaître les informations concernant l'ELEMENT INTERFACE.

Elément_Activé() renvoie un Booléen.

Cette méthode permet de savoir si l'élément de l'interface est activé ou non. Le Booléen est égal à "Vrai" s'il est activé, il est égal à "Faux" sinon.

Responsabilités Privées: /

Classe: **Bouton_Contrôle.**

Super-Classe(s): Elément_Interface.

Sous-Classe(s): Aucun.

*Graphe Hiérarchique:*Elément_Interface.

Description: Cette classe représente un BOUTON (une icône activable au moyen de la souris) tel qu'il peut être affiché à ECRAN, dans une FENETRE ou dans un MENU de l'INTERFACE UTILISATEUR. Il sera activé au moyen de la SOURIS en plaçant le POINTEUR de celle-ci sur le BOUTON et en pressant un des boutons de la SOURIS.

Contrats:

Connaître et manipuler les informations concernant le BOUTON.
Connaître et manipuler l'état d'activation du BOUTON.
Cette responsabilité est héritée de Elément_Interface.

Connaître les informations concernant le BOUTON.

Pointeur_Souris() renvoie un Booléen.

Cette méthode renvoie un booléen égal à "Vrai" si le POINTEUR de la SOURIS se trouve sur le BOUTON. Il est égal à faux sinon.

Bouton_Droit_Souris_Pressé() renvoie un Booléen.

Cette méthode renvoie un booléen égal à Vrai si le bouton droit de la SOURIS est enfoncé et que le POINTEUR de celle-ci se trouve sur le BOUTON.

Bouton_Gauche_Souris_Pressé() renvoie un Booléen.

Cette méthode renvoie un booléen égal à Vrai si le bouton gauche de la SOURIS est enfoncé et que le POINTEUR de celle-ci se trouve sur le BOUTON.

Gérer l'affichage du Bouton dans un écran.
Gérer l'affichage du BOUTON.

Afficher(Fenêtre, Position).

Cette méthode affiche le BOUTON (dans son état normal) dans la FENETRE donnée à la position donnée.

Afficher(Menu, Position).

Cette méthode affiche le BOUTON (dans son état normal) dans le MENU donné à la position donnée.

Mise_Evidence().

Cette méthode change l'affichage du BOUTON en lui donnant son apparence de mise en évidence.

Gérer l'effacement du BOUTON.

Effacer().

Cette méthode efface la représentation du BOUTON qui est actuellement affichée.

Responsabilités Privées:

- Connaître sa représentation graphique.

Classe: **Pointeur_Souris.**

Super-Classe(s): Elément_Interface.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Pointeur_Souris.

Description: Cette classe représente le POINTEUR de la SOURIS qui se déplace à l'ECRAN en fonction des mouvements de la SOURIS.

Contrats:

Connaître et manipuler les informations concernant le POINTEUR SOURIS.
Mettre à jour la représentation graphique du POINTEUR SOURIS.
Changer_Aspect(Représentation_Graphique).

Cette méthode modifie l'aspect du POINTEUR SOURIS à l'écran en lui en assigant une nouvelle représentation graphique.

Connaître les informations concernant le POINTEUR SOURIS.
Position_Pointeur() renvoie une Coordonnée.
Cette méthode fournit les coordonnées du "Point Chaud" du POINTEUR SOURIS à l'écran.

Manipuler l'affichage du POINTEUR SOURIS à l'ECRAN.
Afficher et Effacer le POINTEUR SOURIS à l'ECRAN.
Cette responsabilité est héritée de Elément_Interface.

Responsabilités Privées: /

Classe: **Fenêtre.**

Super-Classe(s): Elément_Interface.

Sous-Classe(s): Fenêtre_Palette_Boutons, Fenêtre_Confirmation, Fenêtre_Sélection_Texte, Fenêtre_Sélection_Images_BI, Fenêtre_Editeur_Texte, Fenêtre_Case_Mémo.

Graphe Hiérarchique: Elément_Interface.

Description: Cette classe représente une zone rectangulaire indépendante contenue dans un ECRAN donné. Elle possède son propre système de coordonnées indépendant de l'écran sur lequel elle est affichée.

Contrats:

Gérer l'affichage de la fenêtre dans un écran.
Ce contrat est hérité de Elément_Interface.

Connaître et manipuler les informations concernant la FENETRE.
Connaître les informations concernant la FENETRE.
Cette responsabilité est en partie héritée de Elément_Interface.

Pointeur_Souris() renvoie un Booléen.

Utilise: Pointeur_Souris.

Cette méthode renvoie un booléen égal à Vrai si le POINTEUR de la SOURIS se trouve dans la FENETRE. Il est égal à faux sinon.

Bouton_Droit_Souris_Pressé() renvoie un Booléen.

Utilise: Pointeur_Souris.

Cette méthode renvoie un booléen égal à Vrai si le bouton droit de la SOURIS est enfoncé et que le POINTEUR de celle-ci se trouve dans la FENETRE.

Bouton_Gauche_Souris_Pressé() renvoie un Booléen.

Utilise: Pointeur_Souris.

Cette méthode renvoie un booléen égal à Vrai si le bouton gauche de la SOURIS est enfoncé et que le POINTEUR de celle-ci se trouve dans la FENETRE.

Position_Pointeur_Fenêtre renvoie une Position.

Utilise: Pointeur_Souris.

Cette méthode renvoie la position (en coordonnées de la fenêtre) du pointeur de la souris dans la fenêtre.

Responsabilités Privées:

- Découper les parties de l'objet graphique à afficher qui dépassent les dimensions de la FENETRE (clipping).
- Connaître ses dimensions et sa position dans l'écran.

Classe: **Fenêtre_Confirmation.**

Super-Classe(s): Fenêtre.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Elément_Interface.

Description: Cette classe représente un type de FENETRE particulier. Lorsque cette FENETRE est activée elle attend une réponse de la part de l'UTILISATEUR. Aucune autre action n'est possible ailleurs dans l'ECRAN tant qu'aucune réponse n'a été fournie. Cette FENETRE particulière contient une phrase (message à faire passer à l'UTILISATEUR) ainsi que deux BOUTONS (réponse positive "OK" et réponse négative "KO").

Contrats:

Activer la fenêtre confirmation.
Activer la fenêtre confirmation.

> **Confirmer(Position, Message) renvoie un Booléen.**
>
> Cette méthode active la fenêtre de confirmation en l'affichant à la position donnée. Elle affiche le message et attendra une action de l'utilisateur. Elle renvoie un Booléen égal à "Vrai" si l'utilisateur a enfoncé le bouton "OK", il sera égal à faux si le bouton "KO" a été enfoncé.

Responsabilités Privées:

- Connaître ses dimensions.

Classe: **Fenêtre_Sélection_Texte.**

Super-Classe(s): Fenêtre.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Elément_Interface.

Description: Cette classe représente une FENETRE particulière qui permet d'afficher une liste d'informations textuelles et d'en sélectionner une parmi l'ensemble.

Contrats:

Activer la fenêtre sélection texte.
Activer la fenêtre sélection texte.

> **Sélectionner_Texte(Position, Message, Liste_Texte) renvoie un élément de Liste_Texte.**
>
> Cette méthode active la fenêtre de sélection de texte en l'affichant à la position donnée. Elle affiche la liste des chaînes de caractères (Liste_Texte) et permet à l'utilisateur de choisir une chaîne de caractères dans cette liste. Elle renvoie la chaîne de caractères sélectionnée.

Responsabilités Privées:

- Connaître ses dimensions.

Classe: **Fenêtre_Editeur_Texte.**

Super-Classe(s): Fenêtre.

Sous-Classe(s): Aucune.

*Graphe Hiérarchique:*Elément_Interface.

Description: Cette classe représente une FENETRE qui permet la saisie d'un texte (ensemble de caractères). Elle se compose d'une zone d'introduction de texte, d'une zone permettant d'afficher un message, et de deux BOUTONS (confirmation et annulation).

Contrats:

Activer la fenêtre éditeur texte.
Activer la fenêtre éditeur texte.

> **Introduire_Texte(Position, Message) renvoie une chaîne de caractères.**
>
> Cette méthode active la fenêtre d'introduction de texte en l'affichant à la position donnée. Elle permet à l'utilisateur d'introduire une chaîne de caractères via le clavier. Elle renvoie la chaîne de caractères introduite.

Responsabilités Privées:

- Connaître ses dimensions.

Classe: **Fenêtre_Sélection_Images_BI.**

Super-Classe(s): Fenêtre.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Elément_Interface.

Description: Cette classe représente un FENETRE permettant la sélection d'IMAGES qui y sont affichées. Elle possède un BOUTON pour confirmer, un BOUTON pour annuler, un BOUTON pour afficher l'ensemble des IMAGES qui précèdent, et un BOUTON pour afficher l'ensemble des IMAGES qui suivent. Le nombre d'IMAGES qui seront affichées à un moment donné dans cette FENETRE est variable et dépend de la taille de chaque IMAGE. Les IMAGES (délimitées par leur contour rectangulaire) contenues dans cette FENETRE SELECTION IMAGES BI ne pourront jamais se chevaucher.

Contrats:

Activer la fenêtre sélection d'images BI.
Activer la fenêtre sélection d'images BI.

> **Choisir_Image(Position, Message, Liste_Noms_Images_BI) renvoie une Image.**
>
> Cette méthode active la fenêtre de sélection d'image en l'affichant à la position donnée. Elle permet à l'utilisateur de sélectionner une image parmi celle qui lui sont proposées. La fenêtre affiche l'ensemble des images de la BI dont les noms sont contenus dans Liste_Noms_Images_BI. Elle renvoie l'image sélectionnée.

Responsabilités Privées:

- Connaître ses dimensions.

Classe: **Fenêtre_Case_Mémo.**

Super-Classe(s): Fenêtre.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Elément_Interface.

Description: Cette classe représente une FENETRE qui permet l'affichage d'un DESSIN en réduction. Elle possède un BOUTON pour commander l'affichage dans cette même fenêtre du DESSIN précédent dans la PAGE BD (s'il existe) et un BOUTON pour commander l'affichage dans cette même fenêtre du DESSIN suivant dans la PAGE BD (s'il existe).

Contrats:

Activer la fenêtre case mémo.
Activer la fenêtre case mémo.

> **Activer_Case_Mémo(Position, Dessin).**
>
> Cette méthode active la fenêtre case mémo en l'affichant à la position donnée. Elle permet l'affichage d'un dessin en réduction. De plus, il est possible à l'utilisateur de demander l'affichage en réduction des dessins précédents et suivants en "cliquant" sur les boutons appropriés.

Responsabilités Privées:

- Connaître ses dimensions.

Classe: **Fenêtre_Palette_Boutons.**

Super-Classe(s): Fenêtre.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Elément_Interface.

Description: Cette classe représente une FENETRE ne contenant que des BOUTONS. L'ensemble, de longueur variable, de ces BOUTONS couvre la surface complète de la FENETRE.

Contrats:

Activer la fenêtre boutons.
Activer la fenêtre boutons.

> **Activer_Fenêtre_Boutons(Position) renvoie le numéro du Bouton sélectionné.**
>
> Cette méthode active la fenêtre boutons en l'affichant à la position donnée. Elle permet l'affichage de l'ensemble des boutons qu'elle contient. Elle attend une action de l'utilisateur sur un de ses boutons et renvoie le numéro du bouton qui a été "pressé" par l'utilisateur.

Responsabilités Privées:

- Connaître ses dimensions.

Classe: **Gestionnaire_Evènements.**

Super-Classe(s): Aucune.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Gestionnaire_Evènements.

Description: Cette classe représente un GESTIONNAIRE D'EVENEMENTS provenant de l'interface avec l'UTILISATEUR. Il gère une liste FIFO où viendront s'accumuler les différents évènements et où il sera possible de retirer au fur et à mesure chaque évènements en vue de l'analyser.

Contrats:

Manipulation de la liste d'évènements.
Consultation de la liste.

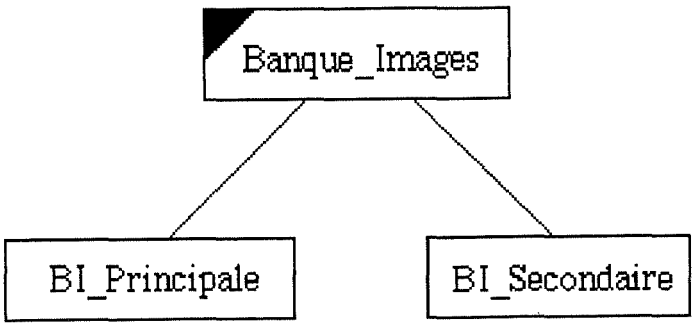
Retirer_Evènement() renvoie un Evènement.
Cette méthode fournit l'évènement le plus ancien de la liste.

Responsabilités Privées:
- Filtrer les évènements selon certains critères.

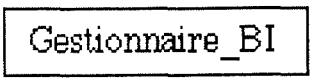
SOUS-SYSTEME GESTION BANQUE D'IMAGES

Ce sous-système encapsule les éléments concernant la représentation et la manipulation des banques d'images.

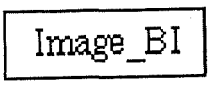
Graphe hiérarchique de Banque d'Images.



Graphe hiérarchique Gestionnaire B.I.



Graphe hiérarchique de Image B.I.



Classe: **Banque_Images.**

Super-Classe(s): Aucune.

Sous-Classe(s): BI_Principale, BI_Secondaire.

Graphe Hiérarchique: Banque_Images

Description: Cette classe représente une BANQUE D'IMAGES qui est un regroupement organisé d'IMAGES_BI (images graphiques sous forme de fichiers) en mémoire secondaire.

Contrats:

Mettre à jour les paramètres de la BANQUE d'IMAGES.

Connaître la racine actuelle.

Get_Racine() renvoie un Chemin_Répertoire.

Cette méthode renvoie le chemin qui indique la racine de la BANQUE d'IMAGES en mémoire secondaire.

Set_Racine(Chemin_Répertoire).

Cette méthode fixe la racine de la BANQUE D'IMAGES à partir du Chemin_Répertoire donné en mémoire secondaire.

Connaître la structure de la B.I.

Get_Structure() renvoie un Arbre.

Cette méthode renvoie la structure arborescente de la B.I. composée de noms de répertoires.

Responsabilités Privées: /

Classe: **BI_Principale.**

Super-Classe(s): Banque_Images.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Banque_Images

Description: Cette classe représente une BANQUE d'IMAGES PRINCIPALE attachée au Logiciel de Création de BD. Il s'agit d'une BANQUE D'IMAGES commune à tous les UTILISATEURS du Logiciel_BD et à partir de laquelle il sera possible d'extraire des IMAGES_BI.

Contrats:

Mettre à jour les paramètres de la BI PRINCIPALE.
Connaître la racine actuelle.

Cette responsabilité est héritée de Banque_Images.

Connaître la structure de la B.I. Principale.

Cette responsabilité est héritée de Banque_Images.

Responsabilités Privées: /

Classe: **BI_Secondaire.**

Super-Classe(s): Banque_Images.

Sous-Classe(s): Aucune.

*Graphe Hiérarchique:*Banque_Images

Description: Cette classe représente une BANQUE d'IMAGES SECONDAIRE attachée à un UTILISATEUR et à lui seul. Cette BI SECONDAIRE sera garnie de fichiers images (IMAGES_BI) provenant uniquement de la BI PRINCIPALE attachée au Logiciel_BD. Elle possédera cependant une organisation qui lui est propre.

Contrats:

Mettre à jour les paramètres de la BI SECONDAIRE.
Connaître la racine actuelle.

Cette responsabilité est héritée de Banque_Images.

Connaître la structure de la B.I. Secondaire.

Cette responsabilité est héritée de Banque_Images.

Connaître l'Utilisateur de la BI Secondaire.

Get_Utilisateur() renvoie un UTILISATEUR.

Cette méthode fournit l'UTILISATEUR auquel appartient la BI SECONDAIRE.

Responsabilités Privées: /

Classe: **Gestionnaire_BI.**

Super-Classe(s): Aucune.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Gestionnaire_BI.

Description: Cette classe représente un gestionnaire de BANQUE d'IMAGES. Il s'occupe de gérer la structure d'une BANQUE d'IMAGES ainsi que les IMAGES_BI qu'elle contient.

Contrats:

Initialiser une Banque d'Images.
Constructeur du Gestionnaire de BANQUE D'IMAGES.

Initialiser(BI_Principale).

Cette méthode crée une occurence de BI Principale.

Initialiser(BI_Secondaire).

Cette méthode crée une occurence de BI Secondaire.

Destructeur de Gestionnaire de BANQUE D'IMAGES.

Détruire(BI_Principale).

Cette méthode détruit l'occurence de BI Principale.

Détruire(BI_Secondaire).

Cette méthode détruit l'occurence de BI Secondaire.

Gestion de la structure d'une BANQUE D'IMAGES.

Modification de l'organisation des Domaines.

Créer_Domaine(BI_Principale, Chemin_Répertoire).

Cette méthode crée un nouveau domaine à l'endroit spécifié par le Chemin_Répertoire donné dans la BI_Principale.

Créer_Domaine(BI_Secondaire, Chemin_Répertoire).

Cette méthode crée un nouveau domaine à l'endroit spécifié par le Chemin_Répertoire donné dans la BI_Secondaire.

Détruire_Domaine(BI_Principale, Chemin_Répertoire, Domaine) renvoie un Booléen.

Cette méthode détruit le Domaine spécifié se trouvant à l'endroit indiqué par le Chemin_Répertoire dans la BI_Principale. Elle renvoie un Booléen égal à "Vrai" si l'opération s'est déroulée normalement. Le Booléen est égal à "Faux" sinon (Domaine non-vide, Domaine inexistant à cet endroit,...).

Détruire_Domaine(BI_Secondaire,Chemin_Répertoire, Domaine) renvoie un Booléen.

Cette méthode détruit le Domaine spécifié se trouvant à l'endroit indiqué par le Chemin_Répertoire dans la BI_Secondaire. Elle renvoie un Booléen égal à "Vrai" si l'opération s'est déroulée normalement. Le Booléen est égal à "Faux" sinon (Domaine non-vide, Domaine inexistant à cet endroit,...).

Gestion du contenu des Domaines.

Modification du contenu des Domaines.

Détruire_Image(BI_Principale, Chemin_Répertoire, Image_BI) renvoie un Booléen.

Cette méthode élimine l'IMAGE_BI se trouvant dans le Domaine spécifié par Chemin_Répertoire dans la BI_Principale. Elle renvoie un Booléen égal à "Vrai" si l'opération s'est déroulée normalement. Le Booléen est égal à "Faux" sinon.

Détruire_Image(BI_Secondaire, Chemin_Répertoire, Image_BI) renvoie un Booléen.

Cette méthode élimine l'IMAGE_BI se trouvant dans le Domaine spécifié par Chemin_Répertoire dans la BI_Secondaire. Elle renvoie un Booléen égal à "Vrai" si l'opération s'est déroulée normalement. Le Booléen est égal à "Faux" sinon.

Déplacer_Image(BI_Principale, Chemin_Répertoire, Chemin_Répertoire, Image_BI) renvoie un Booléen.

Cette méthode déplace l'IMAGE_BI se trouvant dans le Domaine spécifié par le premier Chemin_Répertoire vers le Domaine spécifié par le deuxième Chemin_Répertoire dans la BI_Principale. Elle renvoie un Booléen égal à "Vrai" si l'opération s'est déroulée normalement. Le Booléen est égal à "Faux" sinon.

Déplacer_Image(BI_Secondaire, Chemin_Répertoire, Chemin_Répertoire, Image_BI) renvoie un Booléen.

Cette méthode déplace l'IMAGE_BI se trouvant dans le Domaine spécifié par le premier Chemin_Répertoire vers le Domaine spécifié par le deuxième Chemin_Répertoire dans la BI_Secondaire. Elle renvoie un Booléen égal à "Vrai" si l'opération s'est déroulée normalement. Le Booléen est égal à "Faux" sinon.

Copier_Image(BI_Principale, Chemin_Répertoire, Chemin_Répertoire, Image_BI) renvoie un Booléen.

Cette méthode fait une copie de l'IMAGE_BI se trouvant dans le Domaine spécifié par le premier Chemin_Répertoire vers le Domaine spécifié par le deuxième Chemin_Répertoire dans la BI_Principale. Elle renvoie un Booléen égal à "Vrai" si l'opération s'est déroulée normalement. Le Booléen est égal à "Faux" sinon.

Copier_Image(BI_Secondaire, Chemin_Répertoire, Chemin_Répertoire, Image_BI) renvoie un Booléen.

Cette méthode fait une copie de l'IMAGE_BI se trouvant dans le Domaine spécifié par le premier Chemin_Répertoire vers le Domaine spécifié par le deuxième Chemin_Répertoire dans la BI_Secondaire. Elle renvoie un Booléen égal à "Vrai" si

l'opération s'est déroulée normalement. Le Booléen est égal à "Faux" sinon.

Consultation du contenu des Domaines.

Images_Contenues_Domaine(BI_Principale, Chemin_Répertoire, Domaine) renvoie une liste de noms d'Images_BI.

Cette méthode fournit la liste des IMAGES_BI contenues dans le Domaine spécifié par le Chemin_Répertoire donné de la BI_Principale.

Images_Contenues_Domaine(BI_Secondaire, Chemin_Répertoire, Domaine) renvoie une liste de noms d'Images_BI.

Cette méthode fournit la liste des IMAGES_BI contenues dans le Domaine spécifié par le Chemin_Répertoire donné de la BI_Secondaire.

Domaines_Contenus_Domaine(BI_Principale, Chemin_Répertoire, Domaine) renvoie une liste de noms de Domaines.

Cette méthode fournit la liste des Domaines contenus dans le Domaine spécifié par le Chemin_Répertoire donné de la BI_Principale.

Domaines_Contenus_Domaine(BI_Secondaire, Chemin_Répertoire, Domaine) renvoie une liste de noms de Domaines.

Cette méthode fournit la liste des Domaines contenus dans le Domaine spécifié par le Chemin_Répertoire donné de la BI_Secondaire.

Responsabilités Privées: /

Classe: **Image_BI.**

Super-Classe(s): Aucune.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Image_BI.

Description: Cette classe représente le fichier, en mémoire secondaire, contenant la représentation graphique d'une IMAGE. Elle doit appartenir à une BANQUE D'IMAGES. Son format de codage en mémoire secondaire est un format unique reconnu par le Logiciel_BD.

Contrats:

Fournir des informations concernant l'IMAGE_BI
Fournir des informations concernant l'IMAGE_BI
Format_Codage() renvoie un Format de Codage.

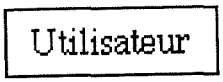
Cette méthode fournit le Format de Codage de l'IMAGE BI en mémoire secondaire.

Responsabilités Privées:
- Connaît son format de codage en mémoire secondaire.

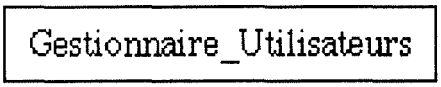
SOUS-SYSTEME GESTION UTILISATEURS

Ce sous-système encapsule les informations et les manipulations concernant les utilisateurs du logiciel de BD.

Graphe hiérarchique de l'Utilisateur.



Graphe hiérarchique de Gestionnaire Utilisateurs.



Classe: **Utilisateur.**

Super-Classe(s): Aucune.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Utilisateur.

Description: Cette classe représente un UTILISATEUR du Logiciel de création de BD. Il s'agit de toute personne (Animateur, Personne Handicapée,...) qui manipule le Logiciel BD et qui se trouve repris dans la liste des UTILISATEURS.

Contrats:

Fournir des informations concernant l'UTILISATEUR.
Fournir des informations concernant l'UTILISATEUR.

Statut() renvoie une Chaîne de Caractères.

Cette méthode fournit le Statut de l'UTILISATEUR.

Get_Nom() renvoie une Chaîne de Caractères.

Cette méthode fournit le Nom de l'Utilisateur.

Responsabilités Privées:
- Connaît son statut (personne handicapée, animateur,...)

Classe: **Gestionnaire_Utilisateurs.**

Super-Classe(s): Aucune.

Sous-Classe(s): Aucune.

Graphe Hiérarchique: Gestionnaire_Utilisateur.

Description: Cette classe représente un gestionnaire de la liste des UTILISATEURS reconnus par le Logiciel BD.

Contrats:

Gestion de la liste des UTILISATEURS.

Modification du contenu de la liste des UTILISATEURS.

Ajouter_Utilisateur(Utilisateur) renvoie un Booléen.

Cette méthode ajoute un nouvel UTILISATEUR à la liste de UTILISATEURS. Elle renvoie un Booléen égal à "Vrai" si l'opération s'est déroulée normalement. Le Booléen est égal à "Faux" sinon (l'UTILISATEUR existe déjà dans la liste).

Retirer_Utilisateur(Utilisateur) renvoie un Booléen.

Cette méthode élimine un UTILISATEURS de la liste des UTILISATEURS. Elle renvoie un Booléen égal à "Vrai" si l'opération s'est déroulée normalement. Le Booléen est égal à "Faux" sinon (l'UTILISATEUR n'existe pas dans la liste).

Consultation de la liste des UTILISATEURS.

Fournir_Liste_Utilisateurs(Type_Utilisateur) renvoie une liste d'Utilisateurs.

Cette méthode fournit l'ensemble de tous les UTILISATEURS de type donné contenus dans la liste des Utilisateurs reconnus par le Logiciel BD. Type_Utilisateur indique s'il s'agit d'un UTILISATEUR handicapé mental ou un animateur.

Responsabilités Privées: /

Les fichiers.

Les fichiers.

Nous présentons dans cette partie les deux grands types de fichiers utilisés dans l'application "Préparation Page BD" du logiciel de création de BD.
Il s'agit de la structure de ces fichiers ainsi que des types des données qui y sont placées.

Fichier "Mise En Page".

Ce fichier contient les informations permettant de reconstituer une Mise En Page.

1. Header.

3 bytes contenant respectivement les caractères 'M', 'E', 'P'.

2. Informations sur la mise en page.

- Hauteur de la page (1 USHORT);
- Largeur de la page (1 USHORT);
- Nombre de cases dans la mise en page (1USHORT).

3. Informations sur les cases.

Ce bloc d'information est répété autant de fois qu'il y a de cases dans la mise en page.

- Position Y (en mm) de la case dans la page (1 USHORT);
- Position X (en mm) de la case dans la page (1 USHORT);
- Hauteur (en mm) de la case (1 USHORT);
- Largeur (en mm) de la case (1 USHORT);

Fichier "Page BD".

Ce fichier contient les informations permettant de reconstituer une Page de BD.

1. Header.

3 bytes contenant respectivement les caractères 'P', 'B', 'D'.

2. Informations sur la mise en page associée à la page de BD.

- Hauteur de la page (1 USHORT);
- Largeur de la page (1 USHORT);
- Nombre de cases dans la mise en page (1USHORT).

3. Informations sur les cases.

Ce bloc d'information est répété autant de fois qu'il y a de cases dans la mise en page associée à la page de BD.

- Position Y (en mm) de la case dans la page (1 USHORT);
- Position X (en mm) de la case dans la page (1 USHORT);
- Hauteur (en mm) de la case (1 USHORT);
- Largeur (en mm) de la case (1 USHORT);

3. Informations sur les dessins.

Ce bloc contient des informations sur les dessins contenus dans la page de BD.

- Nombre de dessins non-vides contenus dans la page de BD (1 USHORT);
- Couleur de fond de tous les dessins de la page de BD (1 BYTE);

3. Informations sur le contenu des dessins.

Ce bloc est répété autant de fois qu'il y a de dessins non-vides dans la page deBD.

- Numéro de la case associée au dessin non-vide(1 USHORT);
- Hauteur (en pixels) du dessin non-vide(1 USHORT);
- Largeur (en pixels) du dessin non-vide(1 USHORT);
- Nombre de plans de bits (profondeur) du dessin non-vide(1 USHORT);
- Contenu de la BitMap du dessin non-vide (n USHORT);

(pour connaître le nombre de USHORT n constituant la BitMap, il faut effectuer le calcul suivant:

n=(((largeur du dessin + 15)/16) * hauteur * profondeur))

Texte du code source.

```
/*
 * ---> Module    : Définition de types de données et de
                    fonctions
 * ---> Type      : Header File
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
 *
*/

#ifndef H_PRIVACY
   #define H_PRIVACY "Définition de types de données et de
                      fonctions"
   #define PRIVATE static
   #define PUBLIC  extern
   typedef enum valbool {False,True} boolean;
#endif
```

```
#define GAD_COUL1 51
#define GAD_COUL2 52
#define GAD_COUL3 53
#define GAD_COUL4 54
#define GAD_COUL5 55
#define GAD_COUL6 56
#define GAD_COUL7 57
#define GAD_COUL8 58
#define GAD_COUL9 59
#define GAD_COUL10 60
#define GAD_COUL11 61
#define GAD_COUL12 62
#define GAD_COUL13 63
#define GAD_COUL14 64
#define GAD_COUL15 65
#define GAD_COUL16 66
```

```
typedef struct
{
   struct Image *Graphique;
   struct Image *OldGraphique;
   struct Window *FenetreImage;
} IMAGE_ACT;

typedef struct
{
   char Texte[80];
   SHORT Cx,Cy;
   SHORT HRad,VRad;
   struct Image Graph;
} BULLE;

typedef struct
{
   char Texte[80];
   RECTANGLE Dim;
   struct Image Graph;
} COMMENTAIRE;

typedef struct
{
   RECTANGLE Dim;
   USHORT NCases;
   NEW_CASE *TabNCase;
} NEW_MEP;

typedef struct
{
   RECTANGLE Dim;
   USHORT NCases;
   struct Window *FenetreMep;
   CASE *Case;
} MISE_EN_PAGE;

typedef struct
{
   MISE_EN_PAGE *Mep;
   USHORT NDessins;
   USHORT NDessinsNNuls;
   DESSIN *Dessin;
   struct Window *FenetrePageBD;
} DOC_PAGEBD;
```

```
/*
 * ---> Module    : Coordinateur des écrans
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include<string.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include<libraries/dos.h>
#include<libraries/diskfont.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_ffond.h"
#include "h_Privacy.h"
#include "h_ObjetsBD.h"
#include "h_Contexte.h"
#include "h_PageBD.h"
#include "h_Ecrans.h"
#include "h_OBulle.h"
#include "h_FConfirm.h"
#include "h_Pointeur.h"


struct IntuitionBase *IntuitionBase;
struct GfxBase       *GfxBase=NULL;
struct Library       *DiskfontBase=NULL;
struct LayersBase    *LayersBase;
struct Screen        *EcranFond;
struct Window        *FenetreFond;
struct RastPort      *FFondRP;
USHORT *pZPointer;
PRIVATE MISE_EN_PAGE *pMep;
PRIVATE int NDessins;
PRIVATE DOC_PAGEBD *pPageBD;
PRIVATE NEW_DESSIN NouvDessins[10];

#define INTUITION_REV 0
#define GRAPHICS_REV 0



PRIVATE void Fermer_Tout_Coordinateur(void);
PRIVATE void Ouvrir_Tout_Coordinateur(void);
PRIVATE USHORT Dialog_Ecran3(void);
PRIVATE USHORT Dialog_Ecran7(void);
PRIVATE int Preparer_PageBD(void);
PRIVATE void Creer_PageBD(void);
```

```
            Fermer_Tout_Coordinateur();
            exit(FALSE);
          }
      FFondRP=FenetreFond->RPort;

      SetRast(FFondRP, 12);

      }


PRIVATE void Fermer_Tout_Coordinateur(void)
      {
        if (FenetreFond)      CloseWindow(FenetreFond);
        if (EcranFond)        CloseScreen(EcranFond);
        if (LayersBase)       CloseLibrary((struct Library *)
LayersBase);
        if (GfxBase)          CloseLibrary(GfxBase);
        if (DiskfontBase)     CloseLibrary(DiskfontBase);
        if (IntuitionBase)    CloseLibrary(IntuitionBase);
      }


PRIVATE USHORT Dialog_Ecran3(void)
{
 USHORT result3;
 USHORT result;

 do
 {
  result3=Ecran3();
  if(result3==1)
  {
   result=Ecran4();
  }
  else if(result3==2)
  {
   result=Ecran5();
  }
 }while(!(result==3) & !(result3==0));
 return(result);
}



PRIVATE int Preparer_PageBD(void)
{
 int reponse=0;
 USHORT result;
 USHORT result1;

 do
 {
  result1=Ecran1();

  switch(result1) {

    case 1: result=Dialog_Ecran3();
            reponse=1;
```

```
   case  2        : result9=Ecran9();
                    break;

   case  1        : result10=Ecran10();
                    if(result10==1)
                    {
                     result8=Dialog_Ecran8();
                    }
                    break;

   case 10        : res=Outil_Bulle_Commentaire('B');
                    break;

   case 11        : res=Outil_Bulle_Commentaire('C');
                    break;

  }
 }while(!(result7==0) && !(result8==0));
 return(0);
}


PRIVATE void Creer_PageBD(void)
{
 USHORT conf=0;
 USHORT result6;
 USHORT result7;

 do
 {
  result6=Ecran6();
  if(result6==1)
  {
   result7=Dialog_Ecran7();
  }
  if(result6==0)
  {
   SetRast(FFondRP, 8);
   conf=Get_FConfirm_Reponse(180,180,"Veux-tu vraiment
sortir?");
  }
  if(result6==2)
  {
   SetRast(FFondRP, 8);
   conf=Get_FConfirm_Reponse(160,180,"Veux-tu vraiment
d0truire la page?");
  }
  if(result6==3)
  {
   SetRast(FFondRP, 8);
   Ecran_Impression_PageBD();
  }
  if(result6==4)
  {
   SetRast(FFondRP, 8);
   Ecran_Sauvetage_PageBD();
  }
 }while(conf!=1);
```

```
/*
 * ---> Module    : ecran1
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include<string.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Gadgets.h"


PRIVATE typedef SHORT boolean;
PRIVATE USHORT selection1;
PRIVATE struct Window *FenetreEcran1;
PRIVATE struct RastPort *FEcran1RastPort;
PUBLIC struct Gadget PageBD, Mep, Stop;
PUBLIC struct Screen *EcranFond;
PUBLIC struct RastPort *FFondRP;
PUBLIC struct TextAttr HELV24, HELV15;



PRIVATE struct NewWindow NewWindowEcran1 = {
     10,10,    /* window XY origin relative to TopLeft of
screen */
     620,492,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETUP, /* IDCMP flags */
     SIMPLE_REFRESH+ACTIVATE+NOCAREREFRESH, /* other window
flags */
     &Mep,     /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     -1,-1,    /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};



PRIVATE void Init_Ecran1(void);
PRIVATE void Detruire_Ecran1(void);
PRIVATE boolean GestionEvent1(struct IntuiMessage *m);
USHORT Ecran1(void);
```

```
        Detruire_Ecran1();
        exit(FALSE);
        }

    FEcran1RastPort = FenetreEcran1->RPort;

    SetAPen(FEcran1RastPort, 1);
    SetDrMd(FEcran1RastPort, JAM1);
    SetRast(FFondRP, 12);
    SetRast(FEcran1RastPort, 10);
    RefreshGList(&Mep, FenetreEcran1, NULL, -1);

    oldTextFont=FEcran1RastPort->Font;

    textFont= (struct TextFont *) OpenDiskFont(&HELV15);
    if(textFont)
    {
     SetFont(FEcran1RastPort, textFont);
     Move(FEcran1RastPort,120,27);
     Text(FEcran1RastPort, "TRAVAIL DANS UNE PAGE DE BANDES
DESSINEES.",42L);
     CloseFont(textFont);
    }

    textFont= (struct TextFont *) OpenDiskFont(&HELV24);
    if(textFont)
    {
     SetFont(FEcran1RastPort, textFont);
     Move(FEcran1RastPort,43,148);
     Text(FEcran1RastPort, "DESIRES-TU : ",13L);

     Move(FEcran1RastPort,192,237);
     Text(FEcran1RastPort, "- Cr0er une NOUVELLE PAGE ?",27L);

     Move(FEcran1RastPort,192,358);
     Text(FEcran1RastPort, "- Continuer une ANCIENNE PAGE
?",31L);


     SetFont(FEcran1RastPort, oldTextFont);
     CloseFont(textFont);
    }

    DrawBorder(FEcran1RastPort,&Soul1,1L,1L);
    DrawBorder(FEcran1RastPort,&TitreBorder,1L,1L);
}


PRIVATE void Detruire_Ecran1(void)
    {
    if (FenetreEcran1)        CloseWindow(FenetreEcran1);
    }




PRIVATE boolean GestionEvent1(m)
```

```
/*
 * ---> Module    : ecran2
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/



#include "stdio.h"
#include "dos.h"
#include<string.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_ObjetsBD.h"
#include "h_Gadgets.h"
#include "h_Privacy.h"
#include "h_Contexte.h"
#include "h_PageBD.h"
#include "h_Misenpage.h"
#include "h_FSeltxt.h"


PRIVATE USHORT selection2, largeur, hauteur, nCases, nonnuls;
PRIVATE NEW_CASE TabCases[20];
PRIVATE NEW_DESSIN TabDessins[20];
PRIVATE DOC_PAGEBD *pPageBD=NULL;
PRIVATE MISE_EN_PAGE *pMep=NULL;
PRIVATE struct Window *FenetreEcran2, *FenetreBoutons2;
PRIVATE struct RastPort *FEcran2RastPort;
PUBLIC struct Gadget Ok, Ko;
PUBLIC struct RastPort *FFondRP;
PUBLIC struct Screen *EcranFond;
PUBLIC struct Window *FenetreFond;
PUBLIC struct TextAttr HELV24, HELV15;
PUBLIC USHORT *pZPointer;



PRIVATE struct NewWindow NewWindowEcran2 = {
     10,10,    /* window XY origin relative to TopLeft of
screen */
     620,492,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETUP, /* IDCMP flags */
     SIMPLE_REFRESH+ACTIVATE+NOCAREREFRESH, /* other window
flags */
     NULL,     /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     -1,-1,    /* maximum width and height */
```

```
    NewWindowEcran2.Width = 620;
    NewWindowEcran2.Height = 492;

    NewWindowEcran2.Screen = EcranFond;

    if (!(FenetreEcran2 = (struct Window *)
OpenWindow(&NewWindowEcran2)))
       {
       printf("La FenΩtre Ecran2 ne peut s'ouvrir ! \n");
       Detr_SelectPageBD2();
       exit(FALSE);
       }

    FEcran2RastPort = FenetreEcran2->RPort;
    SetRast(FEcran2RastPort, 10);
    SetDrMd(FEcran2RastPort, JAM1);


    oldTextFont=FEcran2RastPort->Font;

    textFont= (struct TextFont *) OpenDiskFont(&HELV15);
    if(textFont)
    {
     SetFont(FEcran2RastPort, textFont);
     SetAPen(FEcran2RastPort, 1);
     Move(FEcran2RastPort,120,27);
     Text(FEcran2RastPort, "CONTINUER UNE ANCIENNE PAGE DE
BANDES DESSINEES.",48L);
     CloseFont(textFont);
    }
    DrawBorder(FEcran2RastPort,&TitreBorder,1L,1L);
}


PRIVATE void Detr_SelectPageBD2(void)
{
 if (FenetreEcran2)        CloseWindow(FenetreEcran2);
}


PRIVATE void Init_AffichPageBD2(void)
{
    static struct TextAttr TOPAZ60 = {
         (STRPTR)"topaz.font",
         TOPAZ_SIXTY,0,0
    };

    struct IntuiText IText1 = {
         1,0,JAM1,  /* front and back text pens, drawmode and
fill byte */
         260,500,   /* XY origin relative to container TopLeft
*/
         &TOPAZ60,  /* font pointer or NULL for default */
         "ES-TU SATISFAIT(E) ?",   /* pointer to text */
         NULL  /* next IntuiText structure */
    };
```

```
    RefreshWindowFrame(FenetreBoutons2);
    PrintIText(FEcran2RastPort,&IText1,1L,1L);
    DrawBorder(FEcran2RastPort,&Ligne,1L,1L);
}


PRIVATE void Detr_AffichPageBD2(void)
   {
    if (FenetreBoutons2)       CloseWindow(FenetreBoutons2);
    if (FenetreEcran2)         CloseWindow(FenetreEcran2);
   }


PRIVATE boolean GestionEventAff2(m)
struct IntuiMessage *m;
{
 struct Gadget *g;
 USHORT id;
 boolean retval=FALSE;
 g=(struct Gadget *) m->IAddress;
 id=g->GadgetID;
 switch(id)
 {
  case GAD_OK : Effacer_PageBD(pPageBD);
                Set_Contexte_Mep(pMep);
                Set_Contexte_PageBD(pPageBD);
                retval=TRUE;
                selection2=3;
                break;

  case GAD_KO : Effacer_PageBD(pPageBD);
                Set_Contexte_Mep(NULL);
                Detruire_PageBD(pPageBD);
                Detruire_Mise_En_Page(pMep);
                retval=TRUE;
                selection2=2;
                break;
 }
 return(retval);
}


PRIVATE void Dialog_AffichPageBD2(void)
{
 boolean Sortie=FALSE;
 ULONG class;
 struct IntuiMessage *msg = NULL;



 Init_AffichPageBD2();

 pMep=Init_Mise_En_Page(largeur, hauteur, nCases, TabCases);
 Set_Contexte_Mep(pMep);
 pPageBD=Init_PageBD(pMep, nonnuls, TabDessins);
 ActivateWindow(FenetreFond);
 SetPointer(FenetreFond, pZPointer, 30, 16, -1000, -1000);
 Afficher_PageBD(EcranFond, pPageBD, (float) 1.7, 200, 5);
```

```
/*
 * ---> Module    : ecran3
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include<string.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Gadgets.h"
#include "h_Privacy.h"



PUBLIC struct Gadget Mep, Page, Sortir;
PUBLIC struct Screen *EcranFond;
PUBLIC struct TextAttr HELV24, HELV15;
PUBLIC struct RastPort *FFondRP;
PRIVATE struct Window *FenetreEcran3;
PRIVATE struct RastPort *FEcran3RastPort;
PRIVATE USHORT selection3;


PRIVATE struct NewWindow NewWindowEcran3 = {
     10,10,    /* window XY origin relative to TopLeft of
screen */
     620,492,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETUP, /* IDCMP flags */
     SIMPLE_REFRESH+ACTIVATE+NOCAREREFRESH, /* other window
flags */
     &Page,    /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     -1,-1,    /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};


PRIVATE void Init_Ecran3(void);
PRIVATE void Detruire_Ecran3(void);
PRIVATE boolean GestionEvent3(struct IntuiMessage *m);
USHORT Ecran3(void);
```

```
    exit(FALSE);
   }

   FEcran3RastPort = FenetreEcran3->RPort;

   SetAPen(FEcran3RastPort, 1);
   SetDrMd(FEcran3RastPort, JAM1);

   SetRast(FFondRP, 12);
   SetRast(FEcran3RastPort, 10);
   RefreshGList(&Page, FenetreEcran3, NULL, -1);

   oldTextFont=FEcran3RastPort->Font;

   textFont= (struct TextFont *) OpenDiskFont(&HELV15);
   if(textFont)
   {
    SetFont(FEcran3RastPort, textFont);
    Move(FEcran3RastPort,120,27);
    Text(FEcran3RastPort, "CREATION NOUVELLE PAGE DE BANDES
DESSINEES.",42L);
    CloseFont(textFont);
   }

   textFont= (struct TextFont *) OpenDiskFont(&HELV24);
   if(textFont)
   {
    SetFont(FEcran3RastPort, textFont);
    Move(FEcran3RastPort,20,100);
    Text(FEcran3RastPort, "DESIRES-TU : ",13L);

    Move(FEcran3RastPort,150,175);
    Text(FEcran3RastPort, "- Créer une NOUVELLE",20L);

    Move(FEcran3RastPort,170,205);
    Text(FEcran3RastPort, "DISPOSITION DE CASES ?",22L);

    Move(FEcran3RastPort,150,275);
    Text(FEcran3RastPort, "- Choisir une ANCIENNE",22L);

    Move(FEcran3RastPort,170,305);
    Text(FEcran3RastPort, "DISPOSITION DE CASES ?",22L);


    Move(FEcran3RastPort,150,375);
    Text(FEcran3RastPort, "- Revenir en arrière ?",22L);

    SetFont(FEcran3RastPort, oldTextFont);
    CloseFont(textFont);
   }
   DrawBorder(FEcran3RastPort,&TitreBorder,1L,1L);
   DrawBorder(FEcran3RastPort,&Soul1,1L,1L);
  }
```

```
Detruire_Ecran3();

return(retval);
}
```

```
PRIVATE struct NewWindow NewWindowBoutons4 = {
     0,0, /* window XY origin relative to TopLeft of screen */
     84,512,   /* window width and height */
     0,1, /* detail and block pens */
     GADGETUP, /* IDCMP flags */
     SIMPLE_REFRESH+ACTIVATE+NOCAREREFRESH, /* other window
flags */
     &Ok, /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     -1,-1,    /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};

PRIVATE void Init_Ecran4(void);
PRIVATE void Detruire_Ecran4(void);
PRIVATE boolean GestionEvent4(struct IntuiMessage *m);
PRIVATE void Mep_Suivante(void);
PRIVATE void Mep_Suivante(void);
USHORT Ecran4(void);


PRIVATE void Init_Ecran4(void)
   {

   static struct TextAttr TOPAZ60 = {
        (STRPTR)"topaz.font",
        TOPAZ_SIXTY,0,0
   };

   static struct IntuiText IText1 = {
        1,0,JAM1,   /* front and back text pens, drawmode and
fill byte */
        250,475,    /* XY origin relative to container TopLeft
*/
        &TOPAZ60,   /* font pointer or NULL for default */
        "CHOISIS TA MISE EN PAGE",/* pointer to text */
        NULL   /* next IntuiText structure */
   };

   static SHORT CoordLigne[] = {
    83, 455,
    640, 455
   };

   static struct Border Ligne =
   {
    0,0,
    2,0,
    JAM1,
    2,
    CoordLigne,
    NULL
   };
```

```
PRIVATE void Mep_Suivante(void)
{
 int r;
 USHORT largeur, hauteur, nCases;
 char nom[20];
 NEW_CASE NouvCase[30];

 if((nMep>iMep)&&(iMep>0))
 {
  iMep++;
  Effacer_Mise_En_Page(pMep);
  Detruire_Mise_En_Page(pMep);
  strsfn(tabnom[iMep-1],NULL,NULL,nom,NULL);
  r=Charger_Disque_MEP(drive, chemin, nom, &largeur, &hauteur,
&nCases, NouvCase);
  pMep=Init_Mise_En_Page(largeur, hauteur, nCases, NouvCase);
  Afficher_Mise_En_Page(EcranFond, pMep, taux, 200, 10);
 }
}


PRIVATE void Mep_Precedente(void)
{
 int r;
 USHORT largeur, hauteur, nCases;
 char nom[20];
 NEW_CASE NouvCase[30];

 if((iMep>1)&&(nMep>=iMep))
 {
  iMep--;
  Effacer_Mise_En_Page(pMep);
  Detruire_Mise_En_Page(pMep);
  strsfn(tabnom[iMep-1],NULL,NULL,nom,NULL);
  r=Charger_Disque_MEP(drive, chemin, nom, &largeur, &hauteur,
&nCases, NouvCase);
  pMep=Init_Mise_En_Page(largeur, hauteur, nCases, NouvCase);
  Afficher_Mise_En_Page(EcranFond, pMep, taux, 200, 10);
 }
}



PRIVATE boolean GestionEvent4(m)
struct IntuiMessage *m;
{
 struct Gadget *g;
 USHORT id;
 boolean retval=FALSE;
 g=(struct Gadget *) m->IAddress;
 id=g->GadgetID;
 switch(id)
 {
   case GAD_OK:      retval=TRUE;
                selection4=3;
                Set_Contexte_Mep(pMep);
                Effacer_Mise_En_Page(pMep);
                break;
```

```
        ReplyMsg((struct Message *) msg);
       }
      if(msg=(struct IntuiMessage *)GetMsg(FenetreEcran4-
>UserPort))
      {
       class=msg->Class;
       if(class==GADGETUP)
        Sortie=GestionEvent4(msg);
       ReplyMsg((struct Message *) msg);
      }
     } while(Sortie==FALSE);
    }
    Effacer_Mise_En_Page(pMep);
    retval = selection4;
    Detruire_Ecran4();
    return(retval);
}
```

```
    NULL,     /* window title */
    NULL,     /* custom screen pointer */
    NULL,     /* custom bitmap */
    5,5, /* minimum width and height */
    -1,-1,    /* maximum width and height */
    CUSTOMSCREEN   /* destination screen type */
};


PRIVATE struct NewWindow NewWindowBoutons5 = {
    0,0, /* window XY origin relative to TopLeft of screen */
    84,512,   /* window width and height */
    0,1, /* detail and block pens */
    GADGETUP, /* IDCMP flags */
    SIMPLE_REFRESH+ACTIVATE+NOCAREREFRESH, /* other window
flags */
    &Ok, /* first gadget in gadget list */
    NULL,     /* custom CHECKMARK imagery */
    NULL,     /* window title */
    NULL,     /* custom screen pointer */
    NULL,     /* custom bitmap */
    5,5, /* minimum width and height */
    -1,-1,    /* maximum width and height */
    CUSTOMSCREEN   /* destination screen type */
};


PRIVATE struct NewWindow NewFenetrePage =
 {
   0,0,
   0,0,
   0,1,
   MOUSEBUTTONS,
   REPORTMOUSE|NOCAREREFRESH|RMBTRAP,
   NULL,
   NULL,
   NULL,
   NULL,
   NULL,
   0,0,
   0,0,
   CUSTOMSCREEN
 };



PRIVATE void Init_Ecran5(void);
PRIVATE void Detruire_Ecran5(void);
PRIVATE boolean GestionEvent5(struct IntuiMessage *m);
PRIVATE void Afficher_Page(void);
PRIVATE void Draw_Rectangle(struct RastPort *RPort, USHORT
supx, USHORT supy, USHORT infx, USHORT infy);
PRIVATE void Creer_Case(void);
PRIVATE void Detruire_Case(void);
PRIVATE void Creer_MEP(void);
PRIVATE NEW_CASE Convert_Nouvelle_Case(USHORT supx, USHORT
supy, USHORT infx, USHORT infy);
```

```
   Detr_Case.Flags = GADGIMAGE;
   Disquette.LeftEdge = 4;
   Disquette.TopEdge = 290;
   Disquette.Flags = GADGIMAGE;
   Sortir.LeftEdge = 4;
   Sortir.TopEdge = 370;
   Sortir.Flags = GADGIMAGE;



   NewWindowEcran5.Screen = EcranFond;
   NewWindowBoutons5.Screen = EcranFond;

   if (!(FenetreEcran5 = (struct Window *)
OpenWindow(&NewWindowEcran5)))
      {
      printf("La FenΩtre Ecran5 ne peut s'ouvrir ! \n");
      Detruire_Ecran5();
      exit(FALSE);
      }

   if (!(FenetreBoutons5 = (struct Window *)
OpenWindow(&NewWindowBoutons5)))
      {
      printf("La FenΩtre Boutons5 ne peut s'ouvrir ! \n");
      Detruire_Ecran5();
      exit(FALSE);
      }

   FEcran5RastPort = FenetreEcran5->RPort;
   SetRast(FEcran5RastPort, 10);
   PrintIText(FEcran5RastPort,&IText1,1L,1L);
   DrawBorder(FEcran5RastPort,&Ligne,1L,1L);
   SetRast(FenetreBoutons5->RPort, 8);
   RefreshWindowFrame(FenetreBoutons5);
   RefreshGList(&Ok, FenetreBoutons5, NULL, -1);
}


PRIVATE void Detruire_Ecran5(void)
{
 if (FenetrePage)          CloseWindow(FenetrePage);
 if (FenetreEcran5)        CloseWindow(FenetreEcran5);
 if (FenetreBoutons5)      CloseWindow(FenetreBoutons5);
}


PRIVATE void Afficher_Page(void)
{
 int i;
 struct RastPort *rastPort;

 NewFenetrePage.TopEdge=10;
 NewFenetrePage.LeftEdge=200;
 NewFenetrePage.Width=(SHORT) (LargPage*taux);
 NewFenetrePage.Height=(SHORT) (HautPage*taux);
 NewFenetrePage.MaxWidth=(SHORT) (LargPage*taux);
 NewFenetrePage.MaxHeight=(SHORT) (HautPage*taux);
```

```
 return(NCase);
}


PRIVATE boolean Point_In_Rectangle(USHORT x, USHORT y, RECT
rect)
{
 boolean rep;

 rep=FALSE;
 if((x>=rect.X1)&&(x<=rect.X2)&&(y>=rect.Y1)&&(y<=rect.Y2))
 {
  rep=TRUE;
 }
 return(rep);
}


PRIVATE boolean Case_Chevauche(USHORT x1, USHORT y1, USHORT x2,
USHORT y2)
{
 RECT Case;
 boolean rep;
 int i;

 rep=FALSE;
 Case.X1=x1; Case.Y1=y1; Case.X2=x2; Case.Y2=y2;


 for(i=1;i<=NCases;i++)
 {
  if(Point_In_Rectangle(x1,y1,OldCase[i])){rep=TRUE;}
  if(Point_In_Rectangle(x2,y1,OldCase[i])){rep=TRUE;}
  if(Point_In_Rectangle(x2,y2,OldCase[i])){rep=TRUE;}
  if(Point_In_Rectangle(x1,y2,OldCase[i])){rep=TRUE;}

if(Point_In_Rectangle(OldCase[i].X1,OldCase[i].Y1,Case)){rep=TR
UE;}

if(Point_In_Rectangle(OldCase[i].X2,OldCase[i].Y1,Case)){rep=TR
UE;}

if(Point_In_Rectangle(OldCase[i].X2,OldCase[i].Y2,Case)){rep=TR
UE;}

if(Point_In_Rectangle(OldCase[i].X1,OldCase[i].Y2,Case)){rep=TR
UE;}
 }
 return(rep);
}
```

```
   {
    message=(struct IntuiMessage *) GetMsg(FenetrePage-
>UserPort);
    class=message->Class;
    code=message->Code;
    mousex = message->MouseX;
    mousey = message->MouseY;
    if(class==MOUSEMOVE)
    {
     if((mousex!=oldmx)|(mousey!=oldmy))
     {
      Draw_Rectangle(rastPort, orgx, orgy, oldmx, oldmy);
      Draw_Rectangle(rastPort, orgx, orgy, mousex, mousey);
      oldmx=mousex; oldmy=mousey;
     }
    }
   }

   if(!Case_Chevauche(orgx,orgy,mousex,mousey))
   {
    SetDrMd(rastPort,JAM1);
    SetAPen(rastPort,1);
    SetDrPt(rastPort, 0xCCCC);
    Draw_Rectangle(rastPort, orgx, orgy, mousex, mousey);
    SetDrPt(rastPort, 0xFFFF);
    NCases++;
    OldCase[NCases].X1=orgx;
    OldCase[NCases].Y1=orgy;
    OldCase[NCases].X2=mousex;
    OldCase[NCases].Y2=mousey;
   }
   else
   {
    Draw_Rectangle(rastPort, orgx, orgy, mousex, mousey);
   }

   ReplyMsg((struct Message *) message);
   ModifyIDCMP(FenetrePage, lessFlags);
  }
}


PRIVATE void Draw_Rectangle(struct RastPort *RPort, USHORT
supx, USHORT supy, USHORT infx, USHORT infy)
{
 Move(RPort, supx, supy);
 if(infx<supx) infx=supx;
 if(infy<supy) infy=supy;
 Draw(RPort,infx,supy);
 Move(RPort,infx,supy);
 Draw(RPort,infx,infy);
 Move(RPort,infx,infy);
 Draw(RPort,supx,infy);
 Move(RPort,supx,infy);
 Draw(RPort,supx,supy);
 Move(RPort,supx,supy);
}
```

```
   res=Get_Texte_FIntroTxt(210,170,"Choisis un autre
nom!\0",nom1);
   if(res!=0)
   {
    strmfn(nom2, NULL, NULL, nom1, "MEP");
    existe=Appartient_Disque(nomfich, nom2);
    if(existe) printf("existe dθjα!\n");
   }
  }
  if(!existe)
  {
   Creer_MEP();
   pMep=Get_Contexte_Mep();
   ActivateWindow(FenetreFond);
   SetPointer(FenetreFond, pZPointer, 30, 16, -1000, -1000);
   Sauver_Disque_MEP(pMep, drive, chemin, nom1);
   ClearPointer(FenetreFond);
  }
 }



 Afficher_Page();

 Ok.NextGadget = &Sortir;
 Sortir.NextGadget = &Case;
 Case.NextGadget = &Detr_Case;
 Detr_Case.NextGadget = &Disquette;
 Disquette.NextGadget = NULL;

 Ok.LeftEdge = 4;
 Ok.TopEdge = 50;
 Ok.Flags = GADGIMAGE;
 Case.LeftEdge = 4;
 Case.TopEdge = 130;
 Case.Flags = GADGIMAGE;
 Detr_Case.LeftEdge = 4;
 Detr_Case.TopEdge = 210;
 Detr_Case.Flags = GADGIMAGE;
 Disquette.LeftEdge = 4;
 Disquette.TopEdge = 290;
 Disquette.Flags = GADGIMAGE;
 Sortir.LeftEdge = 4;
 Sortir.TopEdge = 370;
 Sortir.Flags = GADGIMAGE;

 Gadget1=&Ok;
 Position=AddGList(FenetreBoutons5,Gadget1,0,-1,NULL);
 RefreshGList(Gadget1,FenetreBoutons5,NULL,-1);
}


PRIVATE boolean GestionEvent5(m)
struct IntuiMessage *m;
{
 struct Gadget *g;
 USHORT id;
 boolean retval=FALSE;
```

```
    ReplyMsg((struct Message *) message);
    switch(selection5)
    {
     case 0 : break;

     case 1 : Creer_Case();
              break;

     case 2 : Detruire_Case();
              break;

     case 3 : if(NCases>0)
              {
               Creer_MEP();
              }
              else
              {
               Sortie=FALSE;
              }
              break;

     case 4 : if(NCases>0)
              {
               Sauver_Mise_En_Page();
               selection5=3;
              }
              else
              {
               Sortie=FALSE;
              }
              break;

    }
   }
  }while(Sortie==FALSE);
  retval = selection5;
  Detruire_Ecran5();
  return(retval);
}
```

```
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     -1,-1,    /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};


PRIVATE struct NewWindow NewWindowBoutons6 = {
     0,0, /* window XY origin relative to TopLeft of screen */
     84,512,   /* window width and height */
     0,1, /* detail and block pens */
     GADGETUP, /* IDCMP flags */
     SIMPLE_REFRESH+ACTIVATE+NOCAREREFRESH, /* other window
flags */
     &Ok, /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     -1,-1,    /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};

PRIVATE void Init_Ecran6(void);
PRIVATE void Detruire_Ecran6(void);
PRIVATE void FBoutons6_Exclusifs(struct Gadget *g);
PRIVATE void Case_Suivante(void);
PRIVATE void Case_Precedente(void);
PRIVATE boolean GestionEvent6(struct IntuiMessage *m);
USHORT Ecran6(void);
void Ecran_Impression_PageBD(void);
void Ecran_Sauvetage_PageBD(void);


PRIVATE void Init_Ecran6(void)
   {

   static struct TextAttr TOPAZ60 = {
        (STRPTR)"topaz.font",
        TOPAZ_SIXTY,0,0
   };

   static struct IntuiText IText1 = {
        1,0,JAM1,   /* front and back text pens, drawmode and
fill byte */
        250,475,    /* XY origin relative to container TopLeft
*/
        &TOPAZ60,   /* font pointer or NULL for default */
        "CHOISIS TA CASE DE TRAVAIL",   /* pointer to text */
        NULL   /* next IntuiText structure */
   };

   static SHORT CoordLigne[] = {
    83, 455,
    640, 455
   };
```

```
        }

    FEcran6RastPort = FenetreEcran6->RPort;
    SetRast(FEcran6RastPort, 10);
    PrintIText(FEcran6RastPort,&IText1,1L,1L);
    DrawBorder(FEcran6RastPort,&Ligne,1L,1L);
    RefreshGList(&GFlecheG, FenetreEcran6, NULL, -1);
    SetRast(FenetreBoutons6->RPort, 8);
    RefreshWindowFrame(FenetreBoutons6);
    RefreshGList(&Ok, FenetreBoutons6, NULL, -1);
}


PRIVATE void Detruire_Ecran6(void)
    {
    struct Gadget *Gadget1;
    USHORT Position;

    Gadget1=&Ok;
    Position=RemoveGList(FenetreBoutons6,Gadget1,5);
    if (FenetreEcran6)          CloseWindow(FenetreEcran6);
    if (FenetreBoutons6)        CloseWindow(FenetreBoutons6);
    }


PRIVATE void FBoutons6_Exclusifs(struct Gadget *g)
{
 struct Gadget *GSuiv,*Gadget1;
 USHORT Position, GadgetSel;


 GadgetSel=g->GadgetID;
 Gadget1=&Ok;
 Position=RemoveGList(FenetreBoutons6,Gadget1,5);

 Ok.NextGadget = &Sortir;
 Sortir.NextGadget = &Disquette;
 Disquette.NextGadget = &Imprimante;
 Imprimante.NextGadget = &Poubelle;
 Poubelle.NextGadget = NULL;

 Ok.LeftEdge = 4;
 Ok.TopEdge = 20;
 Ok.Flags=GADGIMAGE;
 Disquette.LeftEdge = 4;
 Disquette.TopEdge = 100;
 Disquette.Flags=GADGIMAGE;
 Imprimante.LeftEdge = 4;
 Imprimante.TopEdge = 180;
 Imprimante.Flags=GADGIMAGE;
 Poubelle.LeftEdge = 4;
 Poubelle.TopEdge = 260;
 Poubelle.Flags=GADGIMAGE;
 Sortir.LeftEdge = 4;
 Sortir.TopEdge = 340;
 Sortir.Flags=GADGIMAGE;
```

```
PRIVATE boolean GestionEvent6(m)
struct IntuiMessage *m;
{
 struct Gadget *g;
 USHORT id;
 boolean retval=FALSE;
 g=(struct Gadget *) m->IAddress;
 id=g->GadgetID;
 FBoutons6_Exclusifs(g);
 switch(id)
 {
   case GAD_OK:    retval=TRUE;
                   selection6=1;
                   break;
   case GAD_SORTIR:   retval=TRUE;
                      selection6=0;
                      break;
   case GAD_POUBELLE: retval=TRUE;
                       selection6=2;
                       break;
   case GAD_CASE_PREC: retval=FALSE;
                        Case_Precedente();
                        break;
   case GAD_CASE_SUIV: retval=FALSE;
                        Case_Suivante();
                        break;

   case GAD_IMPRIME:  retval=TRUE;
                      selection6=3;
                      break;

   case GAD_SAUVER:   retval=TRUE;
                      selection6=4;
                      break;

 }
 return(retval);
}



USHORT Ecran6(void)
{
   boolean Sortie=FALSE;
   ULONG class;
   USHORT retval;
   struct IntuiMessage *msg = NULL;

   Init_Ecran6();
   pPageBD6=Get_Contexte_PageBD();
   nCases=Get_Mise_En_Page_Nbre_Cases(pPageBD6->Mep);
   pCaseSel=Get_Contexte_Case_Selectionnee();
   iCase=pCaseSel->Numero;
   Afficher_Mise_En_Page(EcranFond, pPageBD6->Mep, taux, 200,
10);
   do
   {
```

```
  SetPointer(FenetreFond, pZPointer, 30, 16, -1000, -1000);
  print(FenetrePageBD->RPort, FenetrePageBD, 0, 0,
FenetrePageBD->Width, FenetrePageBD->Height);
  ClearPointer(FenetreFond);
 }

 Effacer_PageBD(pPageBD6);
 if (FenetreEcran6)          CloseWindow(FenetreEcran6);

}



void Ecran_Sauvetage_PageBD(void)
{
 USHORT Rep;
 char nom1[20], nom2[20], *chemin, *drive, nomfich[FMSIZE];
 boolean existe;

 NewWindowEcran6.Screen = EcranFond;

 if (!(FenetreEcran6 = (struct Window *)
OpenWindow(&NewWindowEcran6)))
 {
  printf("La FenΩtre Ecran6 ne peut s'ouvrir ! \n");
  CloseWindow(FenetreEcran6);
  exit(FALSE);
 }

 SetRast(FenetreEcran6->RPort, 11);

 pPageBD6=Get_Contexte_PageBD();

 Rep=Get_Texte_FIntroTxt(220, 180, "Quel nom donnes-tu?\0",
nom1);
 if(Rep!=0)
 {
  chemin=Get_Contexte_ZoneTrav();
  drive=Get_Contexte_Drive();
  strmfn(nomfich, drive, chemin, "#?", "PBD");
  strmfn(nom2, NULL, NULL, nom1, "PBD");
  existe=Appartient_Disque(nomfich,nom2);
  if(existe) printf("existe dΘjα!\n");

  while((existe)&&(Rep!=0))
  {
   Rep=Get_Texte_FIntroTxt(220,180,"Choisis un autre
nom!\0",nom1);
   if(Rep!=0)
   {
    strmfn(nom2, NULL, NULL, nom1, "PBD");
    existe=Appartient_Disque(nomfich, nom2);
    if(existe) printf("existe dΘjα!\n");
   }
  }
  if(!existe)
  {
```

```
/*
 * ---> Module    : ecran7
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include<string.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_Gadgets.h"
#include "h_ObjetsBD.h"
#include "h_PageBD.h"
#include "h_FDessin.h"
#include "h_FMemo.h"
#include "h_FOutils1.h"
#include "h_Contexte.h"
#include "h_Dessin.h"
#include "h_OCrayon.h"
#include "h_OGomme.h"
#include "h_OLigne.h"
#include "h_OFill.h"
#include "h_OUndo.h"
#include "h_OPoubelle.h"


PUBLIC struct Gadget FlecheG, FlecheD, Voir, GFlecheG,
GFlecheD, Crayon,
              Gomme, Ligne, Remplir, Bulle, Texte, Ciseaux,
SelImages,
              PageBD, Poubelle, Undo, Sortir;
PUBLIC struct Window *FenetreMemo, *FenetreDessin,
*FenetreOutils1;
PUBLIC struct Screen *EcranFond;
PRIVATE USHORT selection7;
PRIVATE DESSIN *pDessin, *pDessinMemo=NULL;
PRIVATE DOC_PAGEBD *pPBD;
PRIVATE void Init_Ecran7(void);
PRIVATE void Detruire_Ecran7(void);
PRIVATE boolean GestionEvent7(struct IntuiMessage *m);
USHORT Ecran7(USHORT presel);
```

```
  case GAD_CRAYON : selection7=4;
                    retval=False;
                    break;

  case GAD_GOMME : selection7=5;
                   retval=False;
                   break;

  case GAD_LIGNE : selection7=6;
                   retval=False;
                   break;

  case GAD_REMPLIR : selection7=7;
                     retval=False;
                     break;

  case GAD_UNDO : selection7=8;
                  retval=False;
                  break;

  case GAD_POUBELLE : selection7=9;
                      retval=False;
                      break;

  case GAD_BULLE : selection7=10;
                   retval=True;
                   break;

  case GAD_TEXTE : selection7=11;
                   retval=True;
                   break;
 }
 return(retval);
}


USHORT Ecran7(USHORT presel)
   {
   boolean Sortie=False;
   ULONG class;
   USHORT posx, posy;
   RECTANGLE DimFDessin, CoordFDessin;
   struct IntuiMessage *msg = NULL;
   struct Window *FDessin;



   pDessin=Get_Contexte_Dessin_Selectionne();

   Init_Ecran7();


   DimFDessin=Get_FDessin_Dim();
   CoordFDessin=Get_FDessin_Coord();
   posx=(USHORT) (CoordFDessin.Largeur+((DimFDessin.Largeur -
pDessin->Graphique->Width)/2));
```

```
    }
  } while(Sortie==False);

  Effacer_Dessin(pDessin);
  Detruire_Ecran7();

  return(selection7);
}
```

```
PRIVATE boolean GestionEvent8(m)
struct IntuiMessage *m;
{
 struct Gadget *g;
 USHORT id;
 boolean retval=FALSE;
 g=(struct Gadget *) m->IAddress;
 id=g->GadgetID;

 Outils2_Exclusifs(g);
 selection8=100;
 switch(id)
 {
  case GAD_SORTIR : retval=TRUE;
                    selection8=0;
                    break;

  case GAD_COLLE  : retval=TRUE;
                    selection8=1;
                    break;

  case GAD_POUBELLE : retval=FALSE;
                      selection8=2;
                      break;

  case GAD_REDUIRE  : retval=FALSE;
                      selection8=3;
                      break;

  case GAD_CRAYON : retval=FALSE;
                    selection8=4;
                    break;

  case GAD_GOMME : retval=FALSE;
                   selection8=5;
                   break;

  case GAD_LIGNE : retval=FALSE;
                   selection8=6;
                   break;

  case GAD_REMPLIR : retval=FALSE;
                     selection8=7;
                     break;

  case GAD_UNDO : retval=FALSE;
                  selection8=8;
                  break;

  case GAD_ROTATION : retval=FALSE;
                      selection8=9;
                      break;

 }
 return(retval);
}
```

```
                posx=(USHORT)
(CoordFDessin.Largeur+((DimFDessin.Largeur - ImgAct->Graphique-
>Width)/2));
                posy=(USHORT)
(CoordFDessin.Hauteur+(((DimFDessin.Hauteur-45) - ImgAct-
>Graphique->Height)/2));
                FImage=Afficher_Image(ImgAct, EcranFond, posx,
posy);
                Init_FOutils2();
                break;

      case 4 : Detruire_FOutils2();
                ImgAct=(IMAGE_ACT *)
Get_Contexte_Image_Active();
                MAJ_Image_Graphique(ImgAct);
                Outil_Crayon(FImage);
                Init_FOutils2();
                break;

      case 5 : Detruire_FOutils2();
                ImgAct=(IMAGE_ACT *)
Get_Contexte_Image_Active();
                MAJ_Image_Graphique(ImgAct);
                Outil_Gomme(FImage);
                Init_FOutils2();
                break;

      case 6 : Detruire_FOutils2();
                ImgAct=(IMAGE_ACT *)
Get_Contexte_Image_Active();
                MAJ_Image_Graphique(ImgAct);
                Outil_Ligne(FImage);
                Init_FOutils2();
                break;

      case 7 : Detruire_FOutils2();
                ImgAct=(IMAGE_ACT *)
Get_Contexte_Image_Active();
                MAJ_Image_Graphique(ImgAct);
                Outil_Fill(FImage);
                Init_FOutils2();
                break;

      case 8 : ImgAct=(IMAGE_ACT *)
Get_Contexte_Image_Active();
                Effacer_Image(ImgAct);

FImage=Afficher_Image(ImgAct,EcranFond,posx,posy);
                break;

      case 9 : Detruire_FOutils2();
                ImgAct=(IMAGE_ACT *)
Get_Contexte_Image_Active();
                MAJ_Image_Graphique(ImgAct);
                Effacer_Image(ImgAct);
                FImage=NULL;
                Outil_Rotation(ImgAct);
```

```
/*
 * ---> Module    : écran9
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include<string.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_ObjetsBD.h"
#include "h_Gadgets.h"
#include "h_PageBD.h"
#include "h_Contexte.h"



PUBLIC struct Gadget Sortir;
PUBLIC struct Screen *EcranFond;
PUBLIC struct Window *FenetreFond;
PUBLIC struct RastPort *FFondRP;
PUBLIC USHORT *pZPointer;
PRIVATE struct Window *FenetreEcran, *FenetreBoutons;
PRIVATE struct RastPort *FEcranRastPort;
PRIVATE USHORT selection;


PRIVATE struct NewWindow NewWindowEcran = {
     10,10,    /* window XY origin relative to TopLeft of
screen */
     620,492,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETUP, /* IDCMP flags */
     SIMPLE_REFRESH+ACTIVATE+NOCAREREFRESH, /* other window
flags */
     NULL,     /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     -1,-1,    /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};
```

```
    struct Window *OpenWindow();

    Sortir.NextGadget = NULL;

    Sortir.TopEdge=100;
    Sortir.LeftEdge=4;
    Sortir.Flags=GADGIMAGE;

    NewWindowEcran.LeftEdge = 0;
    NewWindowEcran.TopEdge = 0;
    NewWindowEcran.Width = 640;
    NewWindowEcran.Height = 512;

    NewWindowEcran.Screen = EcranFond;
    NewWindowBoutons.Screen = EcranFond;

    NewWindowBoutons.FirstGadget = &Sortir;

    if (!(FenetreEcran = (struct Window *)
OpenWindow(&NewWindowEcran)))
       {
        printf("La FenΩtre Ecran9 ne peut s'ouvrir ! \n");
        Detr_AffichPageBD();
        exit(FALSE);
       }

    if (!(FenetreBoutons = (struct Window *)
OpenWindow(&NewWindowBoutons)))
       {
        printf("La FenΩtre Boutons9 ne peut s'ouvrir ! \n");
        Detr_AffichPageBD();
        exit(FALSE);
       }

    FEcranRastPort = FenetreEcran->RPort;
    SetRast(FEcranRastPort, 10);
    PrintIText(FEcranRastPort,&IText1,1L,1L);
    DrawBorder(FEcranRastPort,&Ligne,1L,1L);
    SetRast(FenetreBoutons->RPort, 8);
    RefreshWindowFrame(FenetreBoutons);
    RefreshGList(&Sortir, FenetreBoutons, NULL, -1);
    }


PRIVATE void Detr_AffichPageBD(void)
   {
    if (FenetreBoutons)       CloseWindow(FenetreBoutons);
    if (FenetreEcran)        CloseWindow(FenetreEcran);
   }
```

```
/*
 * ---> Module    : écran10
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include "dos.h"
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_privacy.h"
#include "h_Gadgets.h"
#include "h_ObjetsBD.h"
#include "h_Contexte.h"
#include "h_Disque.h"
#include "h_FSeltxt.h"
#include "h_FSelimage.h"
#include "h_Disque.h"
#include "h_Image.h"
#include "h_Dessin.h"



PUBLIC struct RastPort *FFondRP;
PRIVATE USHORT selection10;


USHORT Ecran10(void);



USHORT Ecran10(void)
   {
   IMAGE_ACT *pImgAct;
   DESSIN *pDessin;
   struct Image Img;
   int longueur;
   char tabdir[50][20];
   char dirsel[20], message[29], *chemin, *drive, node[FMSIZE],
nomfich[FMSIZE];



   pDessin=Get_Contexte_Dessin_Selectionne();
   Sauver_Dessin_FastRam(pDessin);

   chemin = Get_Contexte_BISec();
   drive = Get_Contexte_Drive();


   strmfn(nomfich, drive, chemin, "#?", NULL);
   strcpy(message,"CHOISIS UN DOMAINE.\0");
```

```
/*
 * ---> Objet     : Fenêtre_Confirmation
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include<string.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_Gadgets.h"



PUBLIC struct Gadget Ok, Ko;
PUBLIC struct Screen *EcranFond;
PUBLIC struct TextAttr HELV15;
PRIVATE struct Window *FenetreConfirm;
PRIVATE struct RastPort *FConfirmRPort;
PRIVATE USHORT Selection;



PRIVATE struct NewWindow NewFConfirm =
{
 0,0,
 300,200,
 0,1,
 GADGETUP,
 SIMPLE_REFRESH|ACTIVATE|NOCAREREFRESH,
 &Ok,
 NULL,
 NULL,
 NULL,
 NULL,
 5,5,
 -1,-1,
 CUSTOMSCREEN,
};



PRIVATE void Init_FConfirm(USHORT posx, USHORT posy, char
*message);
PRIVATE void Detr_FConfirm(void);
PRIVATE boolean Gestion_Event_Confirm(struct IntuiMessage
*msg);
USHORT Get_FConfirm_Reponse(USHORT posx, USHORT posy, char
*message);
```

```
PRIVATE void Detr_FConfirm(void)
{
 struct Gadget *Gadget1;
 USHORT Position;

 Gadget1=&Ok;
 Position=RemoveGList(FenetreConfirm,Gadget1,-1);
 if (FenetreConfirm) CloseWindow(FenetreConfirm);
}



PRIVATE boolean Gestion_Event_Confirm(struct IntuiMessage *msg)
{
 struct Gadget *g;
 USHORT id;
 boolean retval=FALSE;


 g=(struct Gadget *) msg->IAddress;
 id=g->GadgetID;

 switch(id)
 {
   case GAD_OK : retval=TRUE;
                 Selection=1;
                 break;

   case GAD_KO : retval=TRUE;
                 Selection=0;
                 break;
 }
 return(retval);

}



USHORT Get_FConfirm_Reponse(USHORT posx, USHORT posy, char
*message)
{
 boolean Sortie=FALSE;
 ULONG class;
 struct IntuiMessage *msg = NULL;


 Init_FConfirm(posx, posy, message);

 do
   {
    Wait(1L << FenetreConfirm->UserPort->mp_SigBit);

    if(msg=(struct IntuiMessage *)GetMsg(FenetreConfirm-
>UserPort))
     {
      class=msg->Class;
      if(class==GADGETUP)
       Sortie=Gestion_Event_Confirm(msg);
```

```
/*
 * ---> Module    : Fenêtre Dessin
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include<string.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_ObjetsBD.h"


PUBLIC struct Gadget GFlecheG, GFlecheD;
PUBLIC struct Screen *EcranFond;
PRIVATE struct Window *FenetreDessin;
PRIVATE struct RastPort *FDessinRastPort;

PRIVATE struct NewWindow NewWindowDessin = {
     248,0,    /* window XY origin relative to TopLeft of
screen */
     392,512,  /* window width and height */
     0,1, /* detail and block pens */
     CLOSEWINDOW,   /* IDCMP flags */
     SIMPLE_REFRESH+ACTIVATE, /* other window flags */
     &GFlecheG,     /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     "CASE TRAVAIL",     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     -1,-1,    /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};


void Init_FDessin(void);
void Detruire_FDessin(void);
void Refresh_FDessin(void);
RECTANGLE Get_FDessin_Dim(void);
RECTANGLE Get_FDessin_Coord(void);


void Init_FDessin(void)
{

 static SHORT CoordLigne[] = {
  1, 455,
  390, 455
 };
```

```
RECTANGLE Get_FDessin_Coord(void)
{
 RECTANGLE Coord;

 Coord.Hauteur=NewWindowDessin.TopEdge;
 Coord.Largeur=NewWindowDessin.LeftEdge;
 return(Coord);
}


void Refresh_FDessin(void)
{
 static SHORT CoordLigne[] = {
  1, 455,
  390, 455
 };

 static struct Border Ligne =
 {
  0,0,
  2,0,
  JAM1,
  2,
  CoordLigne,
  NULL
 };


 GFlecheG.NextGadget = &GFlecheD;
 GFlecheD.NextGadget = NULL;

 GFlecheG.LeftEdge = 4;
 GFlecheG.TopEdge = 458;
 GFlecheD.LeftEdge = 327;
 GFlecheD.TopEdge = 458;

 SetRast(FenetreDessin->RPort, 10);
 DrawBorder(FenetreDessin->RPort,&Ligne,1L,1L);
 RefreshGList(&GFlecheG, FenetreDessin, NULL, -1);
}
```

```
PRIVATE struct IntuiText GadgetText =
{
  1, 0, JAM2, 46, 80, &HELV15, NULL, NULL,
};


PRIVATE struct Gadget StringGadget =
{
 NULL,
 46,80,
 260,18,
 GADGHCOMP,
 RELVERIFY,
 STRGADGET,
 (APTR)&GadgetBorder,
 NULL,
 &GadgetText,
 NULL,
 (APTR)&StringInfo,
 1,
 NULL,
};


PRIVATE struct NewWindow NewFIntroTxt =
{
 0,0,
 300,200,
 0,1,
 GADGETUP,
 SIMPLE_REFRESH|ACTIVATE|NOCAREREFRESH,
 &StringGadget,
 NULL,
 NULL,
 NULL,
 NULL,
 5,5,
 -1,-1,
 CUSTOMSCREEN,
};



PRIVATE void Init_FIntroTxt(USHORT posx, USHORT posy, char
*message);
PRIVATE void Detr_FIntroTxt(void);
PRIVATE boolean Gestion_Event_Intro(struct IntuiMessage *msg);
USHORT Get_Texte_FIntroTxt(USHORT posx, USHORT posy, char
*message, char *txt);


PRIVATE void Init_FIntroTxt(USHORT posx, USHORT posy, char
*message)
{
 struct TextFont *textFont, *oldTextFont;

 struct Window *OpenWindow();
```

```
PRIVATE boolean Gestion_Event_Intro(struct IntuiMessage *msg)
{
 struct Gadget *g;
 USHORT id;
 boolean retval=FALSE;


 g=(struct Gadget *) msg->IAddress;
 id=g->GadgetID;

 switch(id)
 {
   case GAD_OK : retval=TRUE;
                 Selection=1;
                 break;

   case GAD_SORTIR : retval=TRUE;
                     Selection=0;
                     break;
 }
 return(retval);

}



USHORT Get_Texte_FIntroTxt(USHORT posx, USHORT posy, char
*message, char *txt)
{
 boolean Sortie=FALSE;
 ULONG class;
 struct IntuiMessage *msg = NULL;


 Init_FIntroTxt(posx, posy, message);

 do
   {
    Wait(1L << FenetreIntroTxt->UserPort->mp_SigBit);

    if(msg=(struct IntuiMessage *)GetMsg(FenetreIntroTxt-
>UserPort))
     {
      class=msg->Class;
      if(class==GADGETUP)
       Sortie=Gestion_Event_Intro(msg);
      ReplyMsg((struct Message *) msg);
     }

   }while(Sortie==FALSE);

   strcpy(txt,Buffer);
   Detr_FIntroTxt();
   return(Selection);
}
```

```
    NULL,     /* window title */
    NULL,     /* custom screen pointer */
    NULL,     /* custom bitmap */
    5,5, /* minimum width and height */
    0,0, /* maximum width and height */
    CUSTOMSCREEN   /* destination screen type */
};

void Afficher_FMemo(DESSIN *pDessin);
void Effacer_FMemo(void);


void Afficher_FMemo(DESSIN *pDessin)
{
 float HautDessin, LargDessin, rapport, RLarg, RHaut;
 RECTANGLE ntaille;
 struct Image *Graphique;

 static SHORT CoordLigne[] = {
  1, 155,
  239, 155
 };

 static struct Border Ligne =
 {
  0,0,
  2,0,
  JAM1,
  2,
  CoordLigne,
  NULL
 };

 struct Window *OpenWindow();

 FlecheG.NextGadget = &Voir;
 Voir.NextGadget = &FlecheD;
 FlecheD.NextGadget = NULL;



 Image3.LeftEdge = 0;
 Image3.TopEdge = 0;
 Image3.Width = 32;
 Image3.Height = 28;
 Image3.Depth = 4;
 Image3.ImageData = ImageData3;
 Image3.PlanePick = 0x000F;
 Image3.PlaneOnOff = 0x0000;
 Image3.NextImage = NULL;

 FlecheG.LeftEdge = 2;
 FlecheG.TopEdge = 159;
 FlecheG.Width = 32;
 FlecheG.Height = 28;
 FlecheG.Flags = GADGIMAGE;
 FlecheG.Activation = RELVERIFY;
 FlecheG.GadgetType = BOOLGADGET;
```

```
    SetRast(FMemoDessin->RPort, 0);
    RefreshWindowFrame(FMemoDessin);

    if(pDessin->Graphique->ImageData!=NULL)
    {
     ActivateWindow(FenetreFond);
     SetPointer(FenetreFond, pZPointer, 30, 16, -1000, -1000);
     Graphique=Reduire_Dessin(pDessin, ntaille);
     DrawImage(FMemoDessin->RPort, Graphique, 0, 0);
     free(Graphique->ImageData);
     free(Graphique);
     ClearPointer(FenetreFond);
    }
   }
  }


  void Effacer_FMemo(void)
  {
   struct Gadget *Gadget1;
   USHORT Position;


   if (FMemoDessin)          CloseWindow(FMemoDessin);

   Gadget1=&FlecheG;
   Position=RemoveGList(FenetreMemo, Gadget1, 3);
   if (FenetreMemo)          CloseWindow(FenetreMemo);
  }
```

```
 Bulle.NextGadget = &Texte;
 Texte.NextGadget = &Ciseaux;
 Ciseaux.NextGadget = &SelImages;
 SelImages.NextGadget = &PageBD;
 PageBD.NextGadget = &Poubelle;
 Poubelle.NextGadget = &Undo;
 Undo.NextGadget = &Sortir;
 Sortir.NextGadget = NULL;

 Crayon.LeftEdge = 3;
 Crayon.TopEdge = 14;
 Crayon.Flags=GADGIMAGE;
 Gomme.LeftEdge = 83;
 Gomme.TopEdge = 14;
 Gomme.Flags=GADGIMAGE;
 Ligne.LeftEdge = 162;
 Ligne.TopEdge = 14;
 Ligne.Flags=GADGIMAGE;
 Remplir.LeftEdge = 3;
 Remplir.TopEdge = 90;
 Remplir.Flags=GADGIMAGE;
 Bulle.LeftEdge = 83;
 Bulle.TopEdge = 90;
 Bulle.Flags=GADGIMAGE;
 Texte.LeftEdge = 162;
 Texte.TopEdge = 90;
 Texte.Flags=GADGIMAGE;
 Ciseaux.LeftEdge = 3;
 Ciseaux.TopEdge = 166;
 Ciseaux.Flags=GADGIMAGE;
 SelImages.LeftEdge = 83;
 SelImages.TopEdge = 166;
 SelImages.Flags=GADGIMAGE;
 PageBD.LeftEdge = 162;
 PageBD.TopEdge = 166;
 PageBD.Flags=GADGIMAGE;
 Poubelle.LeftEdge = 3;
 Poubelle.TopEdge = 242;
 Poubelle.Flags=GADGIMAGE;
 Undo.LeftEdge = 83;
 Undo.TopEdge = 242;
 Undo.Flags=GADGIMAGE;
 Sortir.LeftEdge = 162;
 Sortir.TopEdge = 242;
 Sortir.Flags=GADGIMAGE;

 NewWindowOutils1.Screen = EcranFond;

 if (!(FenetreOutils1 = (struct Window *)
OpenWindow(&NewWindowOutils1)))
      {
      printf("La Fenêtre Outils1 ne peut s'ouvrir ! \n");
      Detruire_FOutils1();
      exit(FALSE);
      }
}
```

```
 SelImages.Flags=GADGIMAGE;
 PageBD.LeftEdge = 162;
 PageBD.TopEdge = 166;
 PageBD.Flags=GADGIMAGE;
 Poubelle.LeftEdge = 3;
 Poubelle.TopEdge = 242;
 Poubelle.Flags=GADGIMAGE;
 Undo.LeftEdge = 83;
 Undo.TopEdge = 242;
 Undo.Flags=GADGIMAGE;
 Sortir.LeftEdge = 162;
 Sortir.TopEdge = 242;
 Sortir.Flags=GADGIMAGE;

 Gadget1=&Crayon;
 GSuiv=&Crayon;
 do
 {
  if .(GSuiv->GadgetID==GadgetSel)
  {
   GSuiv->Flags=GADGHCOMP|GADGIMAGE|SELECTED;
  }
  GSuiv=GSuiv->NextGadget;
 }while(!(GSuiv==NULL));
 Position=AddGList(FenetreOutils1,Gadget1,0,12,NULL);
 RefreshGList(Gadget1,FenetreOutils1,NULL,12);
}
```

```
 Reduire.NextGadget = &Rotation;
 Rotation.NextGadget = &Colle;
 Colle.NextGadget = &Poubelle;
 Poubelle.NextGadget = &Undo;
 Undo.NextGadget = &Sortir;
 Sortir.NextGadget = NULL;


 Crayon.LeftEdge = 3;
 Crayon.TopEdge = 14;
 Crayon.Flags=GADGIMAGE;
 Gomme.LeftEdge = 83;
 Gomme.TopEdge = 14;
 Gomme.Flags=GADGIMAGE;
 Ligne.LeftEdge = 162;
 Ligne.TopEdge = 14;
 Ligne.Flags=GADGIMAGE;
 Remplir.LeftEdge = 3;
 Remplir.TopEdge = 90;
 Remplir.Flags=GADGIMAGE;
 Agrandir.LeftEdge = 83;
 Agrandir.TopEdge = 90;
 Agrandir.Flags=GADGIMAGE;
 Reduire.LeftEdge = 162;
 Reduire.TopEdge = 90;
 Reduire.Flags=GADGIMAGE;
 Rotation.LeftEdge = 3;
 Rotation.TopEdge = 166;
 Rotation.Flags=GADGIMAGE;
 Colle.LeftEdge = 83;
 Colle.TopEdge = 166;
 Colle.Flags=GADGIMAGE;
 Poubelle.LeftEdge = 3;
 Poubelle.TopEdge = 242;
 Poubelle.Flags=GADGIMAGE;
 Undo.LeftEdge = 83;
 Undo.TopEdge = 242;
 Undo.Flags=GADGIMAGE;
 Sortir.LeftEdge = 162;
 Sortir.TopEdge = 242;
 Sortir.Flags=GADGIMAGE;

 NewWindowOutils2.Screen = EcranFond;

 if (!(FenetreOutils2 = (struct Window *)
OpenWindow(&NewWindowOutils2)))
    {
     printf("La FenΩtre Outils2 ne peut s'ouvrir ! \n");
     Detruire_FOutils2();
     exit(FALSE);
    }
}


void Detruire_FOutils2(void)
{
 struct Gadget *Gadget1;
 USHORT Position;
```

```
Undo.Flags=GADGIMAGE;
Sortir.LeftEdge = 162;
Sortir.TopEdge = 242;
Sortir.Flags=GADGIMAGE;


Gadget1=&Crayon;
GSuiv=&Crayon;
do
{
 if (GSuiv->GadgetID==GadgetSel)
 {
  GSuiv->Flags=GADGHCOMP|GADGIMAGE|SELECTED;
 }
 GSuiv=GSuiv->NextGadget;
}while(!(GSuiv==NULL));
Position=AddGList(FenetreOutils2,Gadget1,0,11,NULL);
RefreshGList(Gadget1,FenetreOutils2,NULL,11);
}
```

```
PRIVATE struct NewWindow NewWindowSelImages = {
     0,0, /* window XY origin relative to TopLeft of screen */
     640,512,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETUP, /* IDCMP flags */
     SIMPLE_REFRESH+ACTIVATE+NOCAREREFRESH, /* other window
flags */
     &GFlecheG,     /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     -1,-1,    /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};

PRIVATE SHORT CoordLigne[] = {
    83, 455,
    640, 455
   };

PRIVATE struct Border Ligne = {
    0,0,
    2,0,
    JAM1,
    2,
    CoordLigne,
    NULL
   };


PRIVATE void Init_FSelImages(void);
PRIVATE void Detruire_FSelImages(void);
PRIVATE boolean GestionEventSelIm2(struct IntuiMessage *m);
PRIVATE boolean GestionEventSelIm1(struct IntuiMessage *m);
PRIVATE void Liberer_Mem_Image(void);
PRIVATE void Image_Prec(void);
PRIVATE void Image_Suiv(void);
PRIVATE void Cree_Image_Affichable(void);
PRIVATE void Copy_Images(struct Image *dest, struct Image src);
USHORT Selectionne_FSelImages(char tabnom[][20], int longueur,
char chemin[], struct Image *image);



PRIVATE void Copy_Images(struct Image *dest, struct Image src)
{
 dest->LeftEdge=src.LeftEdge;
 dest->TopEdge=src.TopEdge;
 dest->Width=src.Width;
 dest->Height=src.Height;
 dest->Depth=src.Depth;
 dest->ImageData=src.ImageData;
 dest->PlanePick=src.PlanePick;
 dest->PlaneOnOff=src.PlaneOnOff;
 dest->NextImage=NULL;
}
```

```
      printf("La Fenêtre Sélection Images ne peut s'ouvrir !
\n");
      Detruire_FSelImages();
      exit(FALSE);
      }

   if (!(FenetreBoutonsSelIm = (struct Window *)
OpenWindow(&NewWindowBoutonsSelIm)))
      {
      printf("La FenΩtre Boutons ne peut s'ouvrir ! \n");
      Detruire_FSelImages();
      exit(FALSE);
      }

   FSelImagesRastPort = FenetreSelImages->RPort;
   PrintIText(FSelImagesRastPort,&IText1,1L,1L);
   DrawBorder(FSelImagesRastPort,&Ligne,1L,1L);
   SetRast(FenetreBoutonsSelIm->RPort, 8);
   RefreshWindowFrame(FenetreBoutonsSelIm);
   RefreshGList(&Ok, FenetreBoutonsSelIm, NULL, -1);

}


PRIVATE void Detruire_FSelImages(void)
{
 if (FenetreSelImages)        CloseWindow(FenetreSelImages);
 if (FenetreBoutonsSelIm)
CloseWindow(FenetreBoutonsSelIm);
}


PRIVATE void Liberer_Mem_Image(void)
{
 int blargeur=((imaff.Width+15)/16)*2;
 int len=imaff.Depth*blargeur*imaff.Height;
 FreeMem((char *) imaff.ImageData, len);
}


PRIVATE void Image_Prec(void)
{
 boolean iff;
 char fichim[FMSIZE];
 USHORT palette[64];
 WORD scwidth, scheight;
 long viewmode;

 scwidth=0; scheight=0;
 if(numim>0)
 {
  numim--;
  Liberer_Mem_Image();
  SetRast(FSelImagesRastPort, 0);
  RefreshWindowFrame(FenetreSelImages);
  DrawBorder(FSelImagesRastPort,&Ligne,1L,1L);
  RefreshGList(&GFlecheG, FenetreSelImages, NULL, -1);
  strmfp(fichim,path,tnomfich[numim]);
```

```
                    selectionIm=1;
                    break;

  case GAD_SORTIR : retval=TRUE;
                    selectionIm=0;
                    Liberer_Mem_Image();
                    break;
 }
 return(retval);
}


PRIVATE boolean GestionEventSelIm2(m)
struct IntuiMessage *m;
{
 struct Gadget *g;
 USHORT id;
 boolean retval=FALSE;
 g=(struct Gadget *) m->IAddress;
 id=g->GadgetID;
 switch(id)
 {
  case GAD_IM_PREC  :  retval=FALSE;
                       Image_Prec();
                       break;

  case GAD_IM_SUIV : retval=FALSE;
                     Image_Suiv();
                     break;
 }
 return(retval);
}


USHORT Selectionne_FSelImages(char tabnom[][20], int longueur,
char chemin[], struct Image *image)
{
 boolean Sortie=FALSE, iff;
 int i;
 ULONG class;
 USHORT palette[64];
 WORD scwidth, scheight;
 long viewmode;
 char fichim[FMSIZE];
 struct IntuiMessage *msg = NULL;

 numim=0;
 scwidth=0; scheight=0;
 viewmode=0L;
 ltab=longueur;
 strcpy(path,chemin);
 for(i=0;i<ltab;i++)
 {
  strcpy(tnomfich[i],tabnom[i]);
 }

 Init_FSelImages();
```

```
/*
 * ---> Objet     : Fenêtre_Sélection_Texte
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include<string.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_Gadgets.h"

PUBLIC struct Gadget Ok, Ko, Slider, Sortir;
PUBLIC struct PropInfo SliderProp;
PUBLIC struct Screen *EcranFond;
PRIVATE struct Window *FenetreSelection, *FenetreViewListe;
PRIVATE struct RastPort *FSelectRastPort, *FViewListRastPort;
PRIVATE unsigned char EltChoisiBuffer[20];
PRIVATE unsigned char UndoBuffer[20];
PRIVATE UBYTE Buffer[10][20];
PRIVATE USHORT Selection;

PRIVATE struct NewWindow NewWindowSelection = {
     160,80,   /* window XY origin relative to TopLeft of
screen */
     300,360,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETUP, /* IDCMP flags */
     SIMPLE_REFRESH+ACTIVATE+NOCAREREFRESH, /* other window
flags */
     &Slider,  /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     -1,-1,    /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};


PRIVATE struct NewWindow NewWindowViewListe = {
     178,113,  /* window XY origin relative to TopLeft of
screen */
     218,208,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETUP, /* IDCMP flags */
     SIMPLE_REFRESH+ACTIVATE+NOCAREREFRESH, /* other window
flags */
     NULL,     /* first gadget in gadget list */
```

```
PRIVATE struct IntuiText EltChoisiText =
{
 1,0,
 JAM2,
 -110,1,
 NULL,
 (UBYTE *)"Tu as choisi: ",
 NULL
};


PRIVATE SHORT StringPairs[] =
{
 0,0,160,0
};

PRIVATE struct Border StringBorder =
{
 0,9,1,0,JAM1,2,StringPairs,NULL
};

PRIVATE struct Gadget EltChoisiGad =
{
 NULL,
 80,150,
 210,8,
 GADGHCOMP,
 RELVERIFY,
 STRGADGET,
 (APTR)&StringBorder,
 NULL,
 &EltChoisiText,
 NULL,
 (APTR)&EltChoisiInfo,
 200,
 NULL,
};



PRIVATE boolean GestionEventView(struct IntuiMessage *m);
PRIVATE void Make_Liste_Elements(struct Gadget StructGadget[]);
PRIVATE void Sel_Element(USHORT id);
PRIVATE void Detr_Selecteur(void);
USHORT Selectionne_FSelTxt(char tabnom[][20], int longueur,
char *message, char nomsel[20]);


PRIVATE void Make_Liste_Elements(StructGadget)
struct Gadget StructGadget[];
{
 int i,j;
 UBYTE msg[20];

 for (i=0;i<10;i++)
 {
  StructGadget[i]=ElementG;
  EltView[i]=ViewMessage;
```

```
 Sortir.Flags=GADGIMAGE;
 Slider.TopEdge=33;
 Slider.LeftEdge=240;
 Slider.Flags=GADGIMAGE;
 EltChoisiGad.TopEdge=255;
 EltChoisiGad.LeftEdge=130;


 NewWindowSelection.Screen = EcranFond;
 NewWindowViewListe.Screen = EcranFond;

 NewWindowViewListe.FirstGadget = &EltListeG[0];


 if (!(FenetreSelection = (struct Window *)
OpenWindow(&NewWindowSelection)))
 {
  printf("La FenΩtre Selection ne peut s'ouvrir ! \n");
  Detr_Selecteur();
  exit(FALSE);
 }

 if (!(FenetreViewListe = (struct Window *)
OpenWindow(&NewWindowViewListe)))
 {
  printf("La FenΩtre ViewList ne peut s'ouvrir ! \n");
  Detr_Selecteur();
  exit(FALSE);
 }

 FSelectRastPort = FenetreSelection->RPort;
 PrintIText(FSelectRastPort,&IText1,1L,1L);
}


PRIVATE void Detr_Selecteur(void)
{
 struct Gadget *Gadget1;
 USHORT Position;

 if (FenetreSelection)
 {
  Gadget1=&Slider;
  Position=RemoveGList(FenetreSelection,Gadget1,4);
  CloseWindow(FenetreSelection);
 };

 if (FenetreViewListe)
 {
  Gadget1=&EltListeG[0];
  Position=RemoveGList(FenetreViewListe,Gadget1,10);
  CloseWindow(FenetreViewListe);
 };
}
```

```
    }
   }
   Make_Liste_Elements(EltListeG);
   NewWindowViewListe.Screen = EcranFond;
   NewWindowViewListe.FirstGadget = &EltListeG[0];
   if (!(FenetreViewListe = (struct Window *)
OpenWindow(&NewWindowViewListe)))
   {
    printf("La Fenêtre ViewList ne peut s'ouvrir ! \n");
    Detr_Selecteur();
    exit(FALSE);
   }
  }
}


PRIVATE boolean GestionEventView(m)
struct IntuiMessage *m;
{
 struct Gadget *g;
 USHORT id;
 boolean retval=FALSE;

 g=(struct Gadget *) m->IAddress;
 id=g->GadgetID;
 Sel_Element(id);
 return(retval);
}


PRIVATE boolean GestionEventSel(m, longueur, tabnom)
struct IntuiMessage *m;
int longueur;
char tabnom[][20];
{
 struct Gadget *g;
 USHORT id, position;
 boolean retval=FALSE;


 g=(struct Gadget *) m->IAddress;
 id=g->GadgetID;

 switch(id)
 {
   case GAD_OK : retval=TRUE;
                 Selection=1;
                 break;

   case GAD_SORTIR : retval=TRUE;
                     Selection=0;
                     strcpy(EltChoisiBuffer,"
\0");
                     break;
```

```
/*
 * ---> Objet     : Outil_Bulle_Commentaire
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include "dos.h"
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_Gadgets.h"
#include "h_ObjetsBD.h"
#include "h_Dessin.h"
#include "h_Contexte.h"
#include "h_Disque.h"
#include "h_FDessin.h"
#include "h_FIntrotxt.h"
#include "h_Disque.h"
#include "h_Image.h"



PUBLIC struct Gadget Ok,Ko;
PUBLIC struct Screen *EcranFond;
PUBLIC struct TextAttr VENICE;
PUBLIC struct RastPort *FFondRP;
PRIVATE struct Window *FConfirm, *FDessin;
PRIVATE struct RastPort *FConfirmRP, *FDessinRP;
PRIVATE struct TmpRas tmpRas;
PRIVATE DESSIN *pDessin;
PRIVATE USHORT reponse;
PRIVATE WORD areabuffer[200];
PRIVATE struct AreaInfo areaInfo;
PRIVATE PLANEPTR planeptr = NULL;
PRIVATE struct TextFont *textFont;


PRIVATE struct NewWindow NewWindowConfirm = {
     0,192,    /* window XY origin relative to TopLeft of
screen */
     242,320,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETDOWN+GADGETUP,     /* IDCMP flags */
     SIMPLE_REFRESH+NOCAREREFRESH, /* other window flags */
     &Ok, /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
```

```
 Ok.Flags = GADGHCOMP|GADGIMAGE;
 Ko.Flags = GADGHCOMP|GADGIMAGE;


 NewWindowConfirm.Screen = EcranFond;

 if (!(FConfirm = (struct Window *)
OpenWindow(&NewWindowConfirm)))
 {
  printf("La FenΩtre Confirmation ne peut s'ouvrir ! \n");
  Detruire_Ecran_Bulle();
  exit(FALSE);
 }
 FConfirmRP=FConfirm->RPort;

 Afficher_FMemo(NULL);
 Init_FDessin();

 CoordFDessin=Get_FDessin_Coord();
 DimFDessin=Get_FDessin_Dim();
 posx=(USHORT) (CoordFDessin.Largeur+((DimFDessin.Largeur -
pDessin->Graphique->Width)/2));
 posy=(USHORT) (CoordFDessin.Hauteur+(((DimFDessin.Hauteur-45)
- pDessin->Graphique->Height)/2));

 NewFDessin.LeftEdge=posx;
 NewFDessin.TopEdge=posy;
 NewFDessin.Width=pDessin->Graphique->Width;
 NewFDessin.Height=pDessin->Graphique->Height;
 NewFDessin.Screen=EcranFond;

 if (!(FDessin = (struct Window *) OpenWindow(&NewFDessin)))
 {
  printf("La FenΩtre du dessin ne peut s'ouvrir ! \n");
  Detruire_Ecran_Bulle();
  exit(FALSE);
 }

 FDessinRP=FDessin->RPort;

 planeptr = (PLANEPTR) AllocRaster(FDessin->Width,FDessin-
>Height);
 if(planeptr)
 {
  for (i=0; i<200; i++)
       areabuffer[i] = 0;

  InitArea(&areaInfo, areabuffer, (200*2)/5);
  FDessinRP->AreaInfo = &areaInfo;

  InitTmpRas(&tmpRas, planeptr, RASSIZE(FDessin->Width,
FDessin->Height));
  FDessinRP->TmpRas=&tmpRas;
 }

 Refresh_Dessin(pDessin, FDessin);

 textFont= (struct TextFont *) OpenDiskFont(&VENICE);
```

```
  for(i=0;i<bitMap->Depth;i++)
  {
   if(bitMap->Planes[i])
      FreeMem(bitMap->Planes[i],rastersize);
  }
  FreeMem(bitMap, sizeof(struct BitMap));
 }
}



PRIVATE void Split_Texte(char *source, char *ligne1, char
*ligne2)
{
 int erreur;

 erreur=strmid(source,ligne1,1,12);

 if((strlen(source))>12)
 {
  strcat(ligne1,"-");
  erreur=strmid(source,ligne2,13,12);
 }
 if((strlen(source))<=12)
 {
  strcpy(ligne2,"\0");
 }
}



PRIVATE void Draw_Bulle(struct RastPort *RastPort, SHORT posx,
SHORT posy,char *texte)
{
 LONG erreur;
 SHORT l;
 char lgn1[14], lgn2[14];


 SetAPen(RastPort, 0);
 SetOPen(RastPort, 1);
 AreaMove(RastPort,posx-40,posy);
 AreaDraw(RastPort,posx,posy+65);
 AreaDraw(RastPort,posx+40,posy);
 AreaDraw(RastPort,posx-40,posy);
 AreaEnd(RastPort);
 erreur=AreaEllipse(RastPort, posx, posy, 110, 50);
 erreur=AreaEnd(RastPort);
 SetAPen(RastPort, 1);
 DrawEllipse(RastPort, posx, posy, 110, 50);
 Split_Texte(texte,lgn1, lgn2);
 l=TextLength(RastPort, texte, strlen(lgn1));
 Move(RastPort,posx-(l/2),posy-8);
 Text(RastPort, lgn1,(SHORT) strlen(lgn1));
 l=TextLength(RastPort, texte, strlen(lgn2));
 Move(RastPort,posx-(l/2),posy+22);
 Text(RastPort, lgn2, (SHORT) strlen(lgn2));
}
```

```
    code=message->Code;
    while(code!=SELECTUP)
    {
     message=(struct IntuiMessage *) GetMsg(FDessin->UserPort);
     class=message->Class;
     code=message->Code;
     mousex = message->MouseX;
     mousey = message->MouseY;
     if(class==MOUSEMOVE)
     {
      if((mousex!=oldmx)|(mousey!=oldmy))
      {
       if(sorte=='B')
       {
        Draw_Bulle(FDessin->RPort, (SHORT) oldmx, (SHORT)
oldmy, texte);
        Draw_Bulle(FDessin->RPort, (SHORT) mousex, (SHORT)
mousey, texte);
       }
       if(sorte=='C')
       {
        Draw_Commentaire(FDessin->RPort, (SHORT) oldmx, (SHORT)
oldmy, texte);
        Draw_Commentaire(FDessin->RPort, (SHORT) mousex,
(SHORT) mousey, texte);
       }
       oldmx=mousex; oldmy=mousey;
      }
     }
    }
    SetDrMd(FDessin->RPort,JAM1);
    if(sorte=='B')
    {
     Draw_Bulle(FDessin->RPort, (SHORT) mousex, (SHORT) mousey,
texte);
    }
    if(sorte=='C')
    {
     Draw_Commentaire(FDessin->RPort, (SHORT) mousex, (SHORT)
mousey, texte);
    }
    ReplyMsg((struct Message *) message);
   }
  for(i=1;i<100;i++)
  {
   message=(struct IntuiMessage *) GetMsg(FDessin->UserPort);
  }
  RefreshWindowFrame(FDessin);

}
```

```
   Sortie=False;
   do
    {
     Wait((1L << FConfirm->UserPort->mp_SigBit) | (1L <<
FDessin->UserPort->mp_SigBit));
     if(msg=(struct IntuiMessage *)GetMsg(FConfirm->UserPort))
      {
       class=msg->Class;
       if((class==GADGETDOWN)|(class==GADGETUP))
        Sortie=Gestion_Event_Confirm(msg);
       ReplyMsg((struct Message *) msg);
      }
     if(msg=(struct IntuiMessage *)GetMsg(FDessin->UserPort))
      {
       class=msg->Class;
       if(class==MOUSEBUTTONS)
        Gestion_Event_Deplacement(msg, sorte, texte);
      }
    } while(Sortie==FALSE);
   Detruire_Ecran_Bulle();
  }
  else
  {
   reponse=0;
  }

  return(reponse);
 }
```

```
PRIVATE struct NewWindow NewFDessin =
 {
   0,0,
   0,0,
   0,1,
   MOUSEBUTTONS,
   REPORTMOUSE|NOCAREREFRESH|RMBTRAP,
   NULL,
   NULL,
   NULL,
   NULL,
   NULL,
   0,0,
   390,510,
   CUSTOMSCREEN
 };



PRIVATE void Init_FOutil_Collage();
PRIVATE void Detruire_FOutil_Collage();
PRIVATE boolean  Gestion_Event_OCollage(struct IntuiMessage
*msg);
PRIVATE void Gestion_Event_Deplacement(struct IntuiMessage
*msg);
USHORT Outil_Collage(void);


PRIVATE void Init_FOutil_Collage(void)
{
 USHORT posx, posy;
 struct Window *OpenWindow();
 RECTANGLE CoordFDessin, DimFDessin;



 Ok.NextGadget = &Ko;
 Ko.NextGadget = NULL;

 Ok.LeftEdge = 5;
 Ok.TopEdge = 242;
 Ko.LeftEdge = 162;
 Ko.TopEdge = 242;

 Ok.Flags = GADGHCOMP|GADGIMAGE;
 Ko.Flags = GADGHCOMP|GADGIMAGE;


 NewWindowOColl.Screen = EcranFond;

 if (!(FOutilColl = (struct Window *)
OpenWindow(&NewWindowOColl)))
 {
  printf("La FenΩtre Outils Collage ne peut s'ouvrir ! \n");
  Detruire_FOutil_Collage();
  exit(FALSE);
 }
```

```
   bitMap-
>Planes[i]=(PLANEPTR)AllocMem(rastersize,MEMF_CHIP|MEMF_CLEAR|M
EMF_PUBLIC);
    if(bitMap->Planes[i]==NULL)
    {
     free(bitMap->Planes[i]);
     printf("plus de place m@moire pour les bitplanes !!!\n");
     Exit(FALSE);
    }
   }
  pFastImData=Get_Contexte_PFast();
  Mint= 0xC0;
  BltBitMap(FDessin->RPort->BitMap,FDessin->LeftEdge,FDessin-
>TopEdge,bitMap,0,0,FDessin->Width,FDessin-
>Height,Mint,0xFF,NULL);
  blargeur=((Img->Width+15)/16)*2;
  bitmaplen=(Img->Height * blargeur * Img->Depth);
  CopyMem(bitMap->Planes[0], pFastImData,bitmaplen);
  for(i=0;i<bitMap->Depth;i++)
  {
   if(bitMap->Planes[i])
      FreeMem(bitMap->Planes[i],rastersize);
  }
  FreeMem(bitMap, sizeof(struct BitMap));
 }
}


PRIVATE void Gestion_Event_Deplacement(struct IntuiMessage
*msg)
{
   USHORT mousex, mousey;
   ULONG class;
   USHORT code;
   IMAGE_ACT *Img;
   struct RastPort *rastPort;

   rastPort=FDessin->RPort;
   SetWrMsk(rastPort, 0xff);
   class=msg->Class;
   code=msg->Code;
   ReplyMsg((struct Message *) msg);
   if((class==MOUSEBUTTONS)&&(code==SELECTDOWN))
   {
    mousex=msg->MouseX;
    mousey=msg->MouseY;
    Img=Get_Contexte_Image_Active();
    pDessin=Get_Contexte_Dessin_Selectionne();
    Refresh_Dessin(pDessin, FDessin);
    DrawImage(FDessin->RPort, Img->Graphique, mousex, mousey);
    RefreshWindowFrame(FDessin);
   }
}

PRIVATE boolean  Gestion_Event_OCollage(struct IntuiMessage
*msg)
{
  struct Gadget *g;
```

```
/*
 * ---> Objet    : Outil_Crayon
 * ---> Type     : Source complète
 * ---> Auteurs  : Tillieux Marc
                   Pequet Benoît
*/


#include <stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_Gadgets.h"
#include "h_ObjetsBD.h"
#include "h_Contexte.h"
#include "h_Dessin.h"


PUBLIC struct Gadget Couleur1,Couleur2,Couleur3,Couleur4,
                     Couleur5,Couleur6,Couleur7,Couleur8,
                     Couleur9,Couleur10,Couleur11,Couleur12,
                     Couleur13,Couleur14,Couleur15,Couleur16,
                     Crayon1,Crayon2,Crayon3,Sortir;
PUBLIC struct Screen *EcranFond;
PRIVATE struct Window *FOutilCrayon;
PRIVATE struct RastPort *FOutilCrayonRP;


struct NewWindow NewWindowOCrayon = {
     0,192,    /* window XY origin relative to TopLeft of
screen */
     242,320,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETDOWN+GADGETUP,     /* IDCMP flags */
     SIMPLE_REFRESH+NOCAREREFRESH, /* other window flags */
     &Couleur1,     /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     242,320,  /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};


PRIVATE void Init_FOutil_Crayon();
PRIVATE void Detruire_FOutil_Crayon();
PRIVATE void Boutons_Couleurs_Exclusifs(struct Gadget *g);
```

```
 if (!(FOutilCrayon = (struct Window *)
OpenWindow(&NewWindowOCrayon)))
      {
      printf("La Fenêtre Outils Crayon ne peut s'ouvrir ! \n");
      Detruire_FOutil_Crayon();
      exit(FALSE);
      }
 FOutilCrayonRP=FOutilCrayon->RPort;
}



PRIVATE void Detruire_FOutil_Crayon(void)
{
 struct Gadget *Gadget1;
 USHORT Position;

 Gadget1=&Couleur1;
 Position=RemoveGList(FOutilCrayon, Gadget1, 20);
 if (FOutilCrayon)       CloseWindow(FOutilCrayon);
}



PRIVATE void Boutons_Crayons_Exclusifs(struct Gadget *g)
{
 struct Gadget *GSuiv,*Gadget1;
 USHORT Position, idSel;

 idSel=g->GadgetID;

 Gadget1=&Crayon1;
 Position=RemoveGList(FOutilCrayon,Gadget1,3);

 Crayon1.NextGadget = &Crayon2;
 Crayon1.Flags = GADGHCOMP|GADGIMAGE;
 Crayon2.NextGadget = &Crayon3;
 Crayon2.Flags = GADGHCOMP|GADGIMAGE;
 Crayon3.NextGadget = &Sortir;
 Crayon3.Flags = GADGHCOMP|GADGIMAGE;

 Gadget1=&Crayon1;
 GSuiv=&Crayon1;
 if((40<=idSel)&&(idSel<=42))
 {
  do
  {
   if (GSuiv->GadgetID==idSel)
   {
    GSuiv->Flags=GADGHCOMP|SELECTED|GADGIMAGE;
   }
   GSuiv=GSuiv->NextGadget;
  }while(GSuiv!=NULL);
 }
 Position=AddGList(FOutilCrayon,Gadget1,0,3,NULL);
 RefreshGList(Gadget1,FOutilCrayon,NULL,3);
}
```

```
PRIVATE void Gestion_Event_Dessin(struct IntuiMessage *msg,
struct Window *FenCrayon)
{
   PRIVATE USHORT mousex, mousey;
   PRIVATE ULONG class, moreFlags, lessFlags;
   PRIVATE USHORT code;
   PRIVATE int i;
   PRIVATE struct IntuiMessage *message = NULL;
   PRIVATE struct RastPort *rastPort;

   moreFlags=MOUSEBUTTONS|MOUSEMOVE;
   lessFlags=MOUSEBUTTONS;

   rastPort=FenCrayon->RPort;
   class=msg->Class;
   code=msg->Code;
   mousex = msg->MouseX;
   mousey = msg->MouseY;
   ReplyMsg((struct Message *) msg);
   if((class==MOUSEBUTTONS)&&(code==SELECTDOWN))
   {
    ModifyIDCMP(FenCrayon, moreFlags);
    Move(rastPort, mousex, mousey);
    message=(struct IntuiMessage *)GetMsg(FenCrayon->UserPort);
    class=message->Class;
    code=message->Code;
    mousex = message->MouseX;
    mousey = message->MouseY;
    while((class!=MOUSEBUTTONS)&&(code!=SELECTUP))
    {
     if((mousex<FenCrayon->Width)&&(mousey<FenCrayon->Height))
     {
      Draw(rastPort, mousex, mousey);
     }
     message=(struct IntuiMessage *)GetMsg(FenCrayon-
>UserPort);
     class=message->Class;
     code=message->Code;
     mousex=message->MouseX;
     mousey=message->MouseY;
    }
    ReplyMsg((struct Message *) message);
    ModifyIDCMP(FenCrayon, lessFlags);
   }
   for(i=1;i<100;i++)
   {
    message=(struct IntuiMessage *) GetMsg(FenCrayon-
>UserPort);
   }
}


PRIVATE boolean  Gestion_Event_OCrayon(struct IntuiMessage
*msg, struct Window *FenCrayon)
{
  PRIVATE struct Gadget *g;
  PRIVATE USHORT id;
  PRIVATE boolean retval=False;
```

```
                        SetAPen(rastPort, 9);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL11   : retval=False;
                        SetAPen(rastPort, 10);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL12   : retval=False;
                        SetAPen(rastPort, 11);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL13   : retval=False;
                        SetAPen(rastPort, 12);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL14   : retval=False;
                        SetAPen(rastPort, 13);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL15   : retval=False;
                        SetAPen(rastPort, 14);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL16   : retval=False;
                        SetAPen(rastPort, 15);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_CRAYON1  : retval=False;
                         Boutons_Crayons_Exclusifs(g);
                         break;

    case GAD_CRAYON2  : retval=False;
                         Boutons_Crayons_Exclusifs(g);
                         break;

    case GAD_CRAYON3  : retval=False;
                         Boutons_Crayons_Exclusifs(g);
                         break;
  }
  return(retval);
}



void Outil_Crayon(struct Window *FenCrayon)
{
   ULONG class;
   boolean Sortie;
   struct IntuiMessage *msg = NULL;
   struct RastPort *rastPort;
```

```
/*
 * ---> Objet     : Outil_Remplissage
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/



#include <stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_Gadgets.h"
#include "h_ObjetsBD.h"
#include "h_Contexte.h"
#include "h_Dessin.h"


PUBLIC struct Gadget Couleur1,Couleur2,Couleur3,Couleur4,
                     Couleur5,Couleur6,Couleur7,Couleur8,
                     Couleur9,Couleur10,Couleur11,Couleur12,
                     Couleur13,Couleur14,Couleur15,Couleur16,
                     Sortir;
PUBLIC struct Screen *EcranFond;
PRIVATE struct Window *FOutilFill;
PRIVATE struct RastPort *FOutilFillRP;


struct NewWindow NewWindowOFill = {
     0,192,    /* window XY origin relative to TopLeft of
screen */
     242,320,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETDOWN+GADGETUP,     /* IDCMP flags */
     SIMPLE_REFRESH+NOCAREREFRESH, /* other window flags */
     &Couleur1,     /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     242,320,  /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};
```

```
 if (!(FOutilFill = (struct Window *)
OpenWindow(&NewWindowOFill)))
      {
      printf("La Fenêtre Outils Remplir ne peut s'ouvrir !
\n");
      Detruire_FOutil_Fill();
      exit(FALSE);
      }
 FOutilFillRP=FOutilFill->RPort;
}



PRIVATE void Detruire_FOutil_Fill(void)
{
 struct Gadget *Gadget1;
 USHORT Position;

 Gadget1=&Couleur1;
 Position=RemoveGList(FOutilFill, Gadget1, 17);
 if (FOutilFill)       CloseWindow(FOutilFill);
}



PRIVATE void Boutons_Couleurs_Exclusifs(struct Gadget *g)
{
 struct Gadget *GSuiv,*Gadget1;
 USHORT Position, idSel;

 idSel=g->GadgetID;

 Gadget1=&Couleur1;
 Position=RemoveGList(FOutilFill,Gadget1,16);

 Couleur1.NextGadget = &Couleur2;
 Couleur1.Flags = GADGHCOMP|GADGIMAGE;
 Couleur2.NextGadget = &Couleur3;
 Couleur2.Flags = GADGHCOMP|GADGIMAGE;
 Couleur3.NextGadget = &Couleur4;
 Couleur3.Flags = GADGHCOMP|GADGIMAGE;
 Couleur4.NextGadget = &Couleur5;
 Couleur4.Flags = GADGHCOMP|GADGIMAGE;
 Couleur5.NextGadget = &Couleur6;
 Couleur5.Flags = GADGHCOMP|GADGIMAGE;
 Couleur6.NextGadget = &Couleur7;
 Couleur6.Flags = GADGHCOMP|GADGIMAGE;
 Couleur7.NextGadget = &Couleur8;
 Couleur7.Flags = GADGHCOMP|GADGIMAGE;
 Couleur8.NextGadget = &Couleur9;
 Couleur8.Flags = GADGHCOMP|GADGIMAGE;
 Couleur9.NextGadget = &Couleur10;
 Couleur9.Flags = GADGHCOMP|GADGIMAGE;
 Couleur10.NextGadget = &Couleur11;
 Couleur10.Flags = GADGHCOMP|GADGIMAGE;
```

```
PRIVATE boolean  Gestion_Event_OFill(struct IntuiMessage *msg,
struct Window *FenFill)
{
  PRIVATE struct Gadget *g;
  PRIVATE USHORT id;
  PRIVATE boolean retval=False;
  PRIVATE struct RastPort *rastPort;

  g=(struct Gadget *) msg->IAddress;
  id=g->GadgetID;
  rastPort=FenFill->RPort;

  retval=False;
  switch(id)
  {
   case GAD_SORTIR : retval=True;
                     break;

   case GAD_COUL1  : retval=False;
                     SetAPen(rastPort, 0);
                     Set_Contexte_Couleur_Actuelle(0);
                     Boutons_Couleurs_Exclusifs(g);
                     break;

   case GAD_COUL2  : retval=False;
                     SetAPen(rastPort, 1);
                     Set_Contexte_Couleur_Actuelle(1);
                     Boutons_Couleurs_Exclusifs(g);
                     break;

   case GAD_COUL3  : retval=False;
                     SetAPen(rastPort, 2);
                     Set_Contexte_Couleur_Actuelle(2);
                     Boutons_Couleurs_Exclusifs(g);
                     break;

   case GAD_COUL4  : retval=False;
                     SetAPen(rastPort, 3);
                     Set_Contexte_Couleur_Actuelle(3);
                     Boutons_Couleurs_Exclusifs(g);
                     break;

   case GAD_COUL5  : retval=False;
                     SetAPen(rastPort, 4);
                     Set_Contexte_Couleur_Actuelle(4);
                     Boutons_Couleurs_Exclusifs(g);
                     break;

   case GAD_COUL6  : retval=False;
                     SetAPen(rastPort, 5);
                     Set_Contexte_Couleur_Actuelle(5);
                     Boutons_Couleurs_Exclusifs(g);
                     break;

   case GAD_COUL7  : retval=False;
                     SetAPen(rastPort, 6);
                     Set_Contexte_Couleur_Actuelle(6);
                     Boutons_Couleurs_Exclusifs(g);
```

```
void Outil_Fill(struct Window *FenFill)
{
   ULONG class;
   boolean Sortie;
   PLANEPTR planeptr;
   struct TmpRas tmpRas;
   struct IntuiMessage *msg = NULL;
   struct RastPort *rastPort;


   Init_FOutil_Fill();


   rastPort=FenFill->RPort;
   planeptr = (PLANEPTR) AllocRaster(FenFill->Width,FenFill-
>Height);
   if(planeptr)
   {
    InitTmpRas(&tmpRas, planeptr, RASSIZE(FenFill->Width,
FenFill->Height));
    rastPort->TmpRas=&tmpRas;
   }

   SetAPen(rastPort, 2);

   Sortie=False;
   do
   {
    Wait((1L << FOutilFill->UserPort->mp_SigBit) | (1L <<
FenFill->UserPort->mp_SigBit));
    if(msg=(struct IntuiMessage *)GetMsg(FOutilFill->UserPort))
     {
      class=msg->Class;
      if((class==GADGETDOWN)|(class==GADGETUP))
       Sortie=Gestion_Event_OFill(msg, FenFill);
      ReplyMsg((struct Message *) msg);
     }
    if(msg=(struct IntuiMessage *)GetMsg(FenFill->UserPort))
     {
      class=msg->Class;
      if(class==MOUSEBUTTONS)
       Gestion_Event_Dessin(msg, FenFill);
     }
   } while(Sortie==FALSE);

   FreeRaster(planeptr, FenFill->Width, FenFill->Height);
   rastPort->TmpRas=NULL;
   Detruire_FOutil_Fill();
}
```

```
PRIVATE void Gestion_Event_Gomme(struct IntuiMessage *msg,
struct Window *FGomme);
void Outil_Gomme(struct Window *FenGomme);



PRIVATE void Init_FOutil_Gomme(void)
{
 struct Window *OpenWindow();

 Gomme1.NextGadget = &Gomme2;
 Gomme2.NextGadget = &Gomme3;
 Gomme3.NextGadget = &Gomme4;
 Gomme4.NextGadget = &Sortir;
 Sortir.NextGadget = NULL;

 Sortir.LeftEdge = 162;
 Sortir.TopEdge = 242;

 Gomme1.Flags = GADGHCOMP|GADGIMAGE;
 Gomme2.Flags = GADGHCOMP|GADGIMAGE;
 Gomme3.Flags = GADGHCOMP|GADGIMAGE;
 Gomme4.Flags = GADGHCOMP|GADGIMAGE;

 Gomme1.Flags = GADGHCOMP|GADGIMAGE|SELECTED;

 NewWindowOGomme.Screen = EcranFond;

 if (!(FOutilGomme = (struct Window *)
OpenWindow(&NewWindowOGomme)))
      {
      printf("La FenΩtre Outils Gomme ne peut s'ouvrir ! \n");
      Detruire_FOutil_Gomme();
      exit(FALSE);
      }
 FOutilGommeRP=FOutilGomme->RPort;
}



PRIVATE void Detruire_FOutil_Gomme(void)
{
 struct Gadget *Gadget1;
 USHORT Position;


 Gadget1=&Gomme1;
 Position=RemoveGList(FOutilGomme, Gadget1, 5);
 if (FOutilGomme)       CloseWindow(FOutilGomme);
}
```

```
    code=message->Code;
    mousex = message->MouseX;
    mousey = message->MouseY;
    while((class!=MOUSEBUTTONS)&&(code!=SELECTUP))
    {
     if((mousex<FenGomme->Width)&&(mousey<FenGomme->Height))
     {
      Draw(rastPort, mousex, mousey);
     }
     message=(struct IntuiMessage *)GetMsg(FenGomme->UserPort);
     class=message->Class;
     code=message->Code;
     mousex=message->MouseX;
     mousey=message->MouseY;
    }
    ReplyMsg((struct Message *) message);
    ModifyIDCMP(FenGomme, lessFlags);
   }
}
```

```
PRIVATE boolean  Gestion_Event_OGomme(struct IntuiMessage *msg,
struct Window *FenGomme)
{
  PRIVATE struct Gadget *g;
  PRIVATE USHORT id;
  PRIVATE boolean retval=False;
  PRIVATE struct RastPort *rastPort;

  g=(struct Gadget *) msg->IAddress;
  id=g->GadgetID;
  rastPort=FenGomme->RPort;

  retval=False;
  switch(id)
  {
   case GAD_SORTIR : retval=True;
                     break;

   case GAD_GOMME1  : retval=False;
                      Boutons_Gommes_Exclusifs(g);
                      break;

   case GAD_GOMME2  : retval=False;
                      Boutons_Gommes_Exclusifs(g);
                      break;

   case GAD_GOMME3  : retval=False;
                      Boutons_Gommes_Exclusifs(g);
                      break;

   case GAD_GOMME4  : retval=False;
                      Boutons_Gommes_Exclusifs(g);
                      break;
  }
  return(retval);
}
```

```
/*
 * ---> Objet     : Outil_Ligne
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/



#include <stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_Gadgets.h"
#include "h_ObjetsBD.h"
#include "h_Contexte.h"
#include "h_Dessin.h"


PUBLIC struct Gadget Couleur1,Couleur2,Couleur3,Couleur4,
                     Couleur5,Couleur6,Couleur7,Couleur8,
                     Couleur9,Couleur10,Couleur11,Couleur12,
                     Couleur13,Couleur14,Couleur15,Couleur16,
                     Sortir;
PUBLIC struct Screen *EcranFond;
PRIVATE struct Window *FOutilLigne;
PRIVATE struct RastPort *FOutilLigneRP;


PRIVATE struct NewWindow NewWindowOLigne = {
     0,192,    /* window XY origin relative to TopLeft of
screen */
     242,320,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETDOWN+GADGETUP,     /* IDCMP flags */
     SIMPLE_REFRESH+NOCAREREFRESH, /* other window flags */
     &Couleur1,     /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     242,320,  /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};
```

```
 if (!(FOutilLigne = (struct Window *)
OpenWindow(&NewWindowOLigne)))
      {
      printf("La FenΩtre Outils Ligne ne peut s'ouvrir ! \n");
      Detruire_FOutil_Ligne();
      exit(FALSE);
      }
 FOutilLigneRP=FOutilLigne->RPort;
}



PRIVATE void Detruire_FOutil_Ligne(void)
{
 struct Gadget *Gadget1;
 USHORT Position;

 Gadget1=&Couleur1;
 Position=RemoveGList(FOutilLigne, Gadget1, 17);
 if (FOutilLigne)       CloseWindow(FOutilLigne);
}



PRIVATE void Boutons_Couleurs_Exclusifs(struct Gadget *g)
{
 struct Gadget *GSuiv,*Gadget1;
 USHORT Position, idSel;

 idSel=g->GadgetID;

 Gadget1=&Couleur1;
 Position=RemoveGList(FOutilLigne,Gadget1,16);

 Couleur1.NextGadget = &Couleur2;
 Couleur1.Flags = GADGHCOMP|GADGIMAGE;
 Couleur2.NextGadget = &Couleur3;
 Couleur2.Flags = GADGHCOMP|GADGIMAGE;
 Couleur3.NextGadget = &Couleur4;
 Couleur3.Flags = GADGHCOMP|GADGIMAGE;
 Couleur4.NextGadget = &Couleur5;
 Couleur4.Flags = GADGHCOMP|GADGIMAGE;
 Couleur5.NextGadget = &Couleur6;
 Couleur5.Flags = GADGHCOMP|GADGIMAGE;
 Couleur6.NextGadget = &Couleur7;
 Couleur6.Flags = GADGHCOMP|GADGIMAGE;
 Couleur7.NextGadget = &Couleur8;
 Couleur7.Flags = GADGHCOMP|GADGIMAGE;
 Couleur8.NextGadget = &Couleur9;
 Couleur8.Flags = GADGHCOMP|GADGIMAGE;
 Couleur9.NextGadget = &Couleur10;
 Couleur9.Flags = GADGHCOMP|GADGIMAGE;
 Couleur10.NextGadget = &Couleur11;
 Couleur10.Flags = GADGHCOMP|GADGIMAGE;
 Couleur11.NextGadget = &Couleur12;
```

```
      ModifyIDCMP(FenLigne, moreFlags);
      message=(struct IntuiMessage *) GetMsg(FenLigne->UserPort);
      class=message->Class;
      code=message->Code;
      while(code!=SELECTUP)
      {
       message=(struct IntuiMessage *) GetMsg(FenLigne-
>UserPort);
       class=message->Class;
       code=message->Code;
       mousex = message->MouseX;
       mousey = message->MouseY;
       if(class==MOUSEMOVE)
       {
        if((mousex!=oldmx)|(mousey!=oldmy))
        {
         Move(rastPort, orgx, orgy);
         Draw(rastPort, oldmx, oldmy);
         Move(rastPort, orgx, orgy);
         Draw(rastPort, mousex, mousey);
         oldmx=mousex; oldmy=mousey;
        }
       }
      }
      SetAPen(rastPort,CoulAct);
      SetDrMd(rastPort,JAM1);
      Move(rastPort, orgx, orgy);
      Draw(rastPort, mousex, mousey);
      ReplyMsg((struct Message *) message);
      ModifyIDCMP(FenLigne, lessFlags);
     }
    for(i=1;i<100;i++)
    {
     message=(struct IntuiMessage *) GetMsg(FenLigne->UserPort);
    }
  }


  PRIVATE boolean  Gestion_Event_OLigne(struct IntuiMessage *msg,
  struct Window *FenLigne)
  {
    struct Gadget *g;
    USHORT id;
    boolean retval=False;
    struct RastPort *rastPort;

    g=(struct Gadget *) msg->IAddress;
    id=g->GadgetID;
    rastPort=FenLigne->RPort;

    retval=False;
    switch(id)
    {
     case GAD_SORTIR : retval=True;
                       break;

     case GAD_COUL1  : retval=False;
                       SetAPen(rastPort, 0);
```

```
    case GAD_COUL11   : retval=False;
                        SetAPen(rastPort, 10);
                        Set_Contexte_Couleur_Actuelle(10);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL12   : retval=False;
                        SetAPen(rastPort, 11);
                        Set_Contexte_Couleur_Actuelle(11);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL13   : retval=False;
                        SetAPen(rastPort, 12);
                        Set_Contexte_Couleur_Actuelle(12);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL14   : retval=False;
                        SetAPen(rastPort, 13);
                        Set_Contexte_Couleur_Actuelle(13);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL15   : retval=False;
                        SetAPen(rastPort, 14);
                        Set_Contexte_Couleur_Actuelle(14);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

    case GAD_COUL16   : retval=False;
                        SetAPen(rastPort, 15);
                        Set_Contexte_Couleur_Actuelle(15);
                        Boutons_Couleurs_Exclusifs(g);
                        break;

  }
  return(retval);
}



void Outil_Ligne(struct Window *FenLigne)
{
   ULONG class;
   boolean Sortie;
   struct IntuiMessage *msg = NULL;
   struct RastPort *rastPort;


   Init_FOutil_Ligne();


   rastPort=FenLigne->RPort;
   SetAPen(rastPort, 2);

   Sortie=False;
```

```
/*
 * ---> Objet     : Outil_Réduire
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/



#include "stdio.h"
#include "dos.h"
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_Gadgets.h"
#include "h_ObjetsBD.h"
#include "h_Dessin.h"
#include "h_Contexte.h"
#include "h_Disque.h"
#include "h_FDessin.h"
#include "h_FIntrotxt.h"
#include "h_Disque.h"
#include "h_Image.h"
#include "h_ObjAff.h"

PUBLIC struct Gadget Ok,Ko,Reduire;
PUBLIC struct Screen *EcranFond;
PUBLIC struct Window *FenetreFond;
PUBLIC USHORT *pZPointer;
PRIVATE USHORT selection;
PRIVATE UWORD *pdata;
PRIVATE struct Window *FConfirm;
PRIVATE struct RastPort *FConfirmRP, *FImageRP;
PRIVATE struct Image *Graphique;
PRIVATE SHORT RWidth, RHeight, RDepth;

PRIVATE struct NewWindow NewWindowConfirm = {
     0,192,    /* window XY origin relative to TopLeft of
screen */
     242,320,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETDOWN+GADGETUP,     /* IDCMP flags */
     SIMPLE_REFRESH+NOCAREREFRESH, /* other window flags */
     &Ok, /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     242,320,  /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};
```

```
PRIVATE void Draw_Rectangle(struct RastPort *RPort, USHORT
supx, USHORT supy, USHORT infx, USHORT infy)
{
 Move(RPort, supx, supy);
 if(infx<supx) infx=supx;
 if(infy<supy) infy=supy;
 Draw(RPort,infx,supy);
 Move(RPort,infx,supy);
 Draw(RPort,infx,infy);
 Move(RPort,infx,infy);
 Draw(RPort,supx,infy);
 Move(RPort,supx,infy);
 Draw(RPort,supx,supy);
 Move(RPort,supx,supy);
}



PRIVATE void Gestion_Event_Reduction(struct Window *FImage)
{
 USHORT mousex, mousey, oldmx, oldmy, code, reduire;
 ULONG class, moreFlags, lessFlags, bitmaplen, size;
 float facteurx, facteury, Rlarg, Rhaut, Ilarg, Ihaut;
 int blargeur;
 struct IntuiMessage *message = NULL;
 struct RastPort *rastPort;
 IMAGE_ACT *pImage;


 reduire=0;
 moreFlags=MOUSEBUTTONS|MOUSEMOVE;
 lessFlags=MOUSEBUTTONS;
 rastPort=FImage->RPort;


 SetAPen(rastPort,1);
 SetDrMd(rastPort,COMPLEMENT|JAM1);
 pImage=Get_Contexte_Image_Active();
 SetRast(rastPort,0);
 DrawImage(FImage->RPort, pImage->Graphique, 0, 0);

 size=AvailMem(MEMF_CHIP|MEMF_LARGEST);
 printf("Avant Rectangle: %d\n",size);
 Wait(1L << FImage->UserPort->mp_SigBit);
 message=(struct IntuiMessage *)GetMsg(FImage->UserPort);
 class=message->Class;
 code=message->Code;
 ReplyMsg((struct Message *) message);
 if((class==MOUSEBUTTONS)&&(code==SELECTDOWN))
 {
  mousex = message->MouseX;
  mousey = message->MouseY;
  oldmx=mousex; oldmy=mousey;
  Draw_Rectangle(rastPort, 0, 0, mousex, mousey);
  ModifyIDCMP(FImage, moreFlags);
  Wait(1L << FImage->UserPort->mp_SigBit);
  message=(struct IntuiMessage *) GetMsg(FImage->UserPort);
  class=message->Class;
```

```
    }
    Graphique->LeftEdge=0;
    Graphique->TopEdge=0;
    Graphique->Width=RWidth;
    Graphique->Height=RHeight;
    Graphique->Depth=RDepth;
    Graphique->PlanePick=31;
    Graphique->PlaneOnOff=31;
    Graphique->NextImage=NULL;

    Reduire_BitMap(rastPort->BitMap, Graphique, FImage->Width,
FImage->Height, RDepth, FImage->TopEdge, FImage->LeftEdge,
facteurx, facteury);

    SetRast(rastPort, 0);

    DrawImage(FImage->RPort, Graphique, 2, 2);


   }
   SetDrMd(rastPort,JAM1);
   ClearPointer(FenetreFond);
  }
}


PRIVATE boolean  Gestion_Event_Confirm(struct IntuiMessage
*msg)
{
  struct Gadget *g;
  USHORT id;
  boolean retval=False;


  g=(struct Gadget *) msg->IAddress;
  id=g->GadgetID;

  retval=False;
  switch(id)
  {
   case GAD_KO     : retval=True;
                     selection=0;
                     break;


   case GAD_OK     : retval=True;
                     selection=1;
                     break;

   case GAD_REDUIRE : retval=False;
                      selection=2;
                      break;
  }
  return(retval);
}
```

```
      case 2 : Gestion_Event_Reduction(FImage);
               break;

     }
    }

   } while(Sortie==FALSE);

   Detruire_Ecran_Reduire();
   return(selection);
  }
```

```
/*
 * ---> Objet     : Outil_Rotation
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include "stdio.h"
#include "dos.h"
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_Gadgets.h"
#include "h_ObjetsBD.h"
#include "h_Dessin.h"
#include "h_Contexte.h"
#include "h_Disque.h"
#include "h_FDessin.h"
#include "h_FIntrotxt.h"
#include "h_Disque.h"
#include "h_Image.h"
#include "h_ObjAff.h"


PUBLIC struct Gadget Ok,Rot_G,Rot_D;
PUBLIC struct Screen *EcranFond;
PUBLIC struct Window *FenetreFond;
PUBLIC struct Window *FenetreDessin;
PUBLIC USHORT *pZPointer;
PRIVATE USHORT selection, posx, posy;
PRIVATE UWORD *pdata;
PRIVATE struct Window *FConfirm;
PRIVATE struct RastPort *FConfirmRP;
PRIVATE struct Image *Graphique;
PRIVATE SHORT RWidth, RHeight, RDepth;

PRIVATE struct NewWindow NewWindowConfirm = {
     0,192,    /* window XY origin relative to TopLeft of
screen */
     242,320,  /* window width and height */
     0,1, /* detail and block pens */
     GADGETDOWN+GADGETUP,     /* IDCMP flags */
     SIMPLE_REFRESH+NOCAREREFRESH, /* other window flags */
     &Ok, /* first gadget in gadget list */
     NULL,     /* custom CHECKMARK imagery */
     NULL,     /* window title */
     NULL,     /* custom screen pointer */
     NULL,     /* custom bitmap */
     5,5, /* minimum width and height */
     242,320,  /* maximum width and height */
     CUSTOMSCREEN   /* destination screen type */
};
```

```
PRIVATE void Gestion_Rotation_Gauche(IMAGE_ACT *pImage)
{
 LONG size;
 ULONG bitmaplen, len;
 int blargeur;
 struct RastPort *rastPort;
 RECTANGLE CoordFDessin, DimFDessin;


 rastPort=&EcranFond->RastPort;
 DimFDessin=Get_FDessin_Dim();

 if(pImage->Graphique->Height<DimFDessin.Largeur)
 {
  ActivateWindow(FenetreFond);
  SetPointer(FenetreFond, pZPointer, 30, 16, -1000, -1000);

  CoordFDessin=Get_FDessin_Coord();

  RWidth=pImage->Graphique->Height;
  RHeight=pImage->Graphique->Width;
  RDepth=pImage->Graphique->Depth;


  size=AvailMem(MEMF_CHIP|MEMF_LARGEST);
  printf("Avant lib Image : %d\n",size);
  blargeur=((Graphique->Width+15)/16)*2;
  len=Graphique->Depth*blargeur*Graphique->Height;
  if(Graphique->ImageData) FreeMem(Graphique->ImageData, len);
  if(Graphique) FreeMem(Graphique, sizeof(struct Image));
  size=AvailMem(MEMF_CHIP|MEMF_LARGEST);
  printf("AprΦs lib Image : %d\n",size);
  Graphique=(struct Image *) AllocMem(sizeof(struct Image),
MEMF_CHIP);
  blargeur=((RWidth+15)/16)*2;
  bitmaplen=RHeight*blargeur*RDepth;
  if((Graphique->ImageData=(USHORT *)
AllocMem(bitmaplen,MEMF_CHIP|MEMF_CLEAR))==NULL)
  {
   printf("Erreur, manque de Chip Ram\n");
   free(Graphique->ImageData);
   exit(True);
  }
  Graphique->LeftEdge=0;
  Graphique->TopEdge=0;
  Graphique->Width=RWidth;
  Graphique->Height=RHeight;
  Graphique->Depth=RDepth;
  Graphique->PlanePick=31;
  Graphique->PlaneOnOff=31;
  Graphique->NextImage=NULL;

  Rotation_Gauche_90_Image(rastPort->BitMap, Graphique, pImage-
>Graphique->Width, pImage->Graphique->Height, pImage-
>Graphique->Depth, posy+CoordFDessin.Hauteur,
posx+CoordFDessin.Largeur);
```

```
  Graphique->PlanePick=31;
  Graphique->PlaneOnOff=31;
  Graphique->NextImage=NULL;


  Rotation_Droite_90_Image(rastPort->BitMap, Graphique, pImage-
>Graphique->Width, pImage->Graphique->Height, pImage-
>Graphique->Depth, posy+CoordFDessin.Hauteur,
posx+CoordFDessin.Largeur);


  posx=(USHORT) ((DimFDessin.Largeur - pImage->Graphique-
>Width)/2);
  posy=(USHORT) (((DimFDessin.Hauteur-45) - pImage->Graphique-
>Height)/2);

  Refresh_FDessin();

  DrawImage(FenetreDessin->RPort, Graphique, posx, posy);

  ClearPointer(FenetreFond);
 }
}



PRIVATE boolean  Gestion_Event_Confirm(struct IntuiMessage
*msg)
{
  struct Gadget *g;
  USHORT id;
  boolean retval=False;


  g=(struct Gadget *) msg->IAddress;
  id=g->GadgetID;

  retval=False;
  switch(id)
  {

   case GAD_OK : retval=True;
                 selection=1;
                 break;

   case GAD_ROT_G : retval=False;
                    selection=2;
                    break;



   case GAD_ROT_D : retval=False;
                    selection=3;
                    break;
  }
  return(retval);
}
```

```
/*
 * ---> Objet     : Outil_Undo
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/



#include <stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_Privacy.h"
#include "h_Gadgets.h"
#include "h_ObjetsBD.h"
#include "h_Contexte.h"
#include "h_Dessin.h"



void Outil_Undo(struct Window *FenDessin)
{
   DESSIN *pDessin;


   pDessin=Get_Contexte_Dessin_Selectionne();
   Refresh_Dessin(pDessin, FenDessin);
}
```

```
 SHORT tablignes[10];
 USHORT EtatCase;
 int i;

 fhaut = Fenetre->Height;
 flarg = Fenetre->Width;
 phaut = 296; /*DimPage.Hauteur;*/
 plarg = 210; /*DimPage.Largeur;*/


 tablignes[0]=Milli_2_Pix(Case->CoinGauche, flarg, plarg);
 tablignes[1]=Milli_2_Pix(Case->CoinSup, fhaut, phaut);
 tablignes[2]=tablignes[0]+Milli_2_Pix(Case->Dim.Largeur,
flarg, plarg);
 tablignes[3]=tablignes[1];
 tablignes[4]=tablignes[2];
 tablignes[5]=tablignes[1]+Milli_2_Pix(Case->Dim.Hauteur,
fhaut, phaut);
 tablignes[6]=tablignes[0];
 tablignes[7]=tablignes[5];
 tablignes[8]=tablignes[0];
 tablignes[9]=tablignes[1];

 EtatCase=Etat_Select_Case(Case);

 if(EtatCase==0)
 {
  SetDrPt(Fenetre->RPort, 0xFFFF);
  SetAPen(Fenetre->RPort, 1);
  Move(Fenetre->RPort, tablignes[0], tablignes[1]);
  PolyDraw(Fenetre->RPort, 5, &tablignes[0]);
 }
 else if(EtatCase==1)
 {
  SetDrPt(Fenetre->RPort,0xCCCC);
  SetAPen(Fenetre->RPort, 1);

  for(i=1;i<4;i++)
  {
   Move(Fenetre->RPort, tablignes[0], tablignes[1]);
   PolyDraw(Fenetre->RPort, 5, &tablignes[0]);
   tablignes[0]--;
   tablignes[1]--;
   tablignes[2]++;
   tablignes[3]--;
   tablignes[4]++;
   tablignes[5]++;
   tablignes[6]--;
   tablignes[7]++;
   tablignes[8]--;
   tablignes[9]--;
  }
 }
}
```

```
 }
}

void Deplacer_Case(CASE *Case, struct Window *Fenetre,
RECTANGLE DimPage, SHORT dx, SHORT dy)
{

 Effacer_Case(Fenetre, DimPage, Case);

 Case->CoinSup=(USHORT) Case->CoinSup+dy;
 Case->CoinGauche=(USHORT) Case->CoinGauche+dx;

 Afficher_Case(Fenetre, DimPage, Case);
}


void Selectionner_Case(CASE *Case)
{
 Case->EtatSelect=1;
}

void Deselectionner_Case(CASE *Case)
{
 Case->EtatSelect=0;
}

USHORT Etat_Select_Case(CASE *Case)
{
 USHORT EtatCase;

 EtatCase=Case->EtatSelect;
 return(EtatCase);
}


RECTANGLE Dimensions_Case(CASE *Case)
{
 RECTANGLE dim;

 dim.Hauteur=Case->Dim.Hauteur;
 dim.Largeur=Case->Dim.Largeur;
 return(dim);
}
```

```
 t = r>0? m | mask1 : m & (~mask1);
 BM->Planes[0][BYTEpos]=t;

 mask2=2;
 m=(UBYTE) BM->Planes[1][BYTEpos];
 r = mask2 & couleur;
 t = r>0? m | mask1 : m & (~mask1);
 BM->Planes[1][BYTEpos]=t;

 mask2=4;
 m=(UBYTE) BM->Planes[2][BYTEpos];
 r = mask2 & couleur;
 t = r>0? m | mask1 : m & (~mask1);
 BM->Planes[2][BYTEpos]=t;

 mask2=8;
 m=(UBYTE) BM->Planes[3][BYTEpos];
 r = mask2 & couleur;
 t = r>0? m | mask1 : m & (~mask1);
 BM->Planes[3][BYTEpos]=t;

}



void Reduire_BitMap(struct BitMap *BMSource, struct Image
*Graphique, SHORT Width, SHORT Height, SHORT Depth, SHORT
TopEdge, SHORT LeftEdge, float facteurx, float facteury)
{
 UBYTE coul;
 USHORT *pUSHORT;
 LONG result;
 ULONG rastersize, size;
 int i, j, x, y, y2, x2, oldx2, oldy2, Mint, nbreUSHORT;
 struct RastPort *rastPort1;
 struct BitMap *BMResult=NULL;
 SHORT RWidth, RHeight, RDepth;

 Mint=0xC0;



 RDepth=Depth;
 RWidth=(SHORT) (Width * facteurx);
 RHeight=(SHORT) (Height * facteury);


 if(BMResult=(struct BitMap *) AllocMem(sizeof(struct BitMap),
MEMF_PUBLIC|MEMF_CLEAR))
 {
   InitBitMap (BMResult, RDepth, RWidth, RHeight);

   for(i=0;i<RDepth;i++)
   {
    BMResult->Planes[i]=(PLANEPTR) AllocRaster(RWidth,
RHeight);
    if((BMResult->Planes[i])==NULL)
```

```
      FreeRaster(BMResult->Planes[i],RWidth,RHeight);
      BMResult->Planes[i]=NULL;
     }
    }
    FreeMem(BMResult, sizeof(struct BitMap));
    BMResult=NULL;

  }
 }
```

```
 pMep=Get_Contexte_Mep();
 pCase=Get_Mise_En_Page_Case(NouvDessin.NumCase, pMep);
 pDessin = (struct DESSIN *) malloc(sizeof(DESSIN));
 pDessin->NumCase=NouvDessin.NumCase;
 pDessin->CoulFond=Get_Contexte_Couleur_Fond();
 pDessin->FenetreDessin=NULL;

 if(NouvDessin.Graphique==NULL)
 {
   HautCase=(float) pCase->Dim.Hauteur;
   LargCase=(float) pCase->Dim.Largeur;
   rapport=(float) (LargCase/HautCase);
   DimFDessin=Get_FDessin_Dim();
   LargDessin=(float) (DimFDessin.Largeur-20);
   HautDessin=(float) LargDessin / rapport;

   if(HautDessin>(DimFDessin.Hauteur-100))
   {
    HautDessin=(float) (DimFDessin.Hauteur-100);
    LargDessin=(float) HautDessin*rapport;
   }

   LDessin=(short int) LargDessin; HDessin=(short int)
HautDessin;

   pGraph= (struct Image *) malloc(sizeof(struct Image));

   pDessin->Graphique=pGraph;

   pDessin->Graphique->Width=(SHORT) LDessin;
   pDessin->Graphique->Height=(SHORT) HDessin;
   pDessin->Graphique->LeftEdge=0;
   pDessin->Graphique->TopEdge=0;
   pDessin->Graphique->Depth=4;
   pDessin->Graphique->ImageData=NULL;
   pDessin->Graphique->PlanePick=31;
   pDessin->Graphique->PlaneOnOff=31;
   pDessin->Graphique->NextImage=NULL;
 }
 else
 {
   pDessin->Graphique=NouvDessin.Graphique;
 }

 pDessin->NextDessin=NULL;
 return(pDessin);
}




void Detruire_Dessin(DESSIN *pDessin)
{
```

```
    0,0,
    0,0,
    0,1,
    MOUSEBUTTONS,
    REPORTMOUSE|NOCAREREFRESH|RMBTRAP|SUPER_BITMAP,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    0,0,
    0,0,
    CUSTOMSCREEN
  };


  Mint=0xC0;
  Width=pDessin->Graphique->Width;
  Height=pDessin->Graphique->Height;
  Depth=pDessin->Graphique->Depth;

  NewFenetreDessin.LeftEdge=posx;
  NewFenetreDessin.TopEdge=posy;
  NewFenetreDessin.Width=Width;
  NewFenetreDessin.Height=Height;
  NewFenetreDessin.MaxWidth=Width;
  NewFenetreDessin.MaxHeight=Height;
  NewFenetreDessin.Screen=Ecran;

  FenDessin=NULL;
  if(supbitMap=(struct BitMap *)AllocMem(sizeof(struct BitMap),
MEMF_PUBLIC|MEMF_CLEAR))
  {
    InitBitMap(supbitMap,Depth,Width,Height);
    rastersize=supbitMap->BytesPerRow * supbitMap->Rows;
    for(i=0;i<Depth;i++)
    {
     supbitMap-
>Planes[i]=(PLANEPTR)AllocMem(rastersize,MEMF_CHIP|MEMF_CLEAR|M
EMF_PUBLIC);
     if(supbitMap->Planes[i]==NULL)
     {
      free(supbitMap->Planes[i]);
      printf("plus de place m0moire pour les bitplanes !!!\n");
      Exit(FALSE);
     }
    }
    NewFenetreDessin.BitMap=supbitMap;
    if (!(FenDessin = (struct Window *)
OpenWindow(&NewFenetreDessin)))
    {
     printf("Le Dessin ne peut s'afficher! \n");
     Exit(FALSE);
    }
    FenDessin->RPort->Layer->Window=(APTR) FenDessin;

    blargeur=((Width+15)/16)*2;
    bitmaplen=(Height * blargeur * Depth);
```

```
 struct Window *FenetreDessin;
 USHORT *pFastImData;
 SHORT Width, Height, Depth;
 ULONG bitmaplen, rastersize;
 int i,blargeur,Mint;

 Mint=0xC0;
 pFastImData=Get_Contexte_PFast();
 FenetreDessin=Get_Dessin_FenetreDessin(pDessin);
 Width=pDessin->Graphique->Width;
 Height=pDessin->Graphique->Height;
 Depth=pDessin->Graphique->Depth;

 if(FenetreDessin!=NULL)
 {
  rastport=FenetreDessin->RPort;
  supbitMap=FenetreDessin->WLayer->SuperBitMap;
  bitMap=FenetreDessin->RPort->BitMap;
  blargeur=((Width+15)/16)*2;
  bitmaplen=(Height * blargeur * Depth);
  rastersize=supbitMap->BytesPerRow * supbitMap->Rows;
  BltBitMap(bitMap,FenetreDessin->LeftEdge,FenetreDessin-
>TopEdge,supbitMap,0,0,FenetreDessin->Width,FenetreDessin-
>Height,Mint,0xFF,NULL);
  CopyMem(supbitMap->Planes[0], pFastImData,bitmaplen);
  pDessin->Graphique->ImageData=pFastImData;
  CloseWindow(FenetreDessin);
  for(i=0;i<supbitMap->Depth;i++)
  {
   if(supbitMap->Planes[i])
      FreeMem(supbitMap->Planes[i],rastersize);
  }
  FreeMem(supbitMap, sizeof(struct BitMap));
  pDessin->DRPort.BitMap=NULL;
 }
 Set_Dessin_FenetreDessin(pDessin, NULL);
}


void Sauver_Dessin_FastRam(DESSIN *pDessin)
{
 struct BitMap *supbitMap, *bitMap;
 struct RastPort *rastport;
 struct Window *FenetreDessin;
 USHORT *pFastImData, *pUSHORT,i,j;
 SHORT Width, Height, Depth;
 ULONG bitmaplen, rastersize;
 int blargeur,Mint, nbreUSHORT;



 Mint=0xC0;
 pFastImData=Get_Contexte_PFast();
 FenetreDessin=Get_Dessin_FenetreDessin(pDessin);
 Width=pDessin->Graphique->Width;
 Height=pDessin->Graphique->Height;
 Depth=pDessin->Graphique->Depth;
```

```
  pDessin->Graphique->ImageData=Get_Contexte_PFast();
}


void Set_Dessin_FenetreDessin(DESSIN *pDessin, struct Window
*pFDessin)
{
 pDessin->FenetreDessin=pFDessin;
}


struct Window *Get_Dessin_FenetreDessin(DESSIN *pDessin)
{
 return(pDessin->FenetreDessin);
}


struct Image *Reduire_Dessin(DESSIN *pDessin, RECTANGLE
ntaille)
{
 int i, j, nbreUSHORT, blargeur;
 float facteurx, facteury,num, den;
 USHORT *pUSHORT;
 ULONG rastersize, size, bitmaplen;
 SHORT Width, Height, Depth, RWidth, RHeight, RDepth;
 struct Image *Graphique;
 struct BitMap *BM=NULL;



 Width=pDessin->Graphique->Width;
 Height=pDessin->Graphique->Height;
 Depth=pDessin->Graphique->Depth;

 if((ntaille.Largeur<Width)&&(ntaille.Hauteur<Height))
 {
  num=(float) ntaille.Largeur;
  den=(float) Width;
  facteurx=(float) (num/den);
  num=(float) ntaille.Hauteur;
  den=(float) Height;
  facteury=(float) (num/den);


  RDepth=Depth;
  RWidth=(SHORT) (Width * facteurx);
  RHeight=(SHORT) (Height * facteury);

  Graphique=(struct Image *) AllocMem(sizeof(struct Image),
MEMF_CHIP);
  blargeur=((RWidth+15)/16)*2;
  bitmaplen=RHeight*blargeur*RDepth;
  if((Graphique->ImageData=(USHORT *)
AllocMem(bitmaplen,MEMF_CHIP|MEMF_CLEAR))==NULL)
  {
   printf("Erreur, manque de Chip Ram\n");
   free(Graphique->ImageData);
   exit(True);
```

```
    }
   }
   return(Graphique);
  }
```

```
 pImgAct->Graphique->TopEdge=0;
 pImgAct->Graphique->Width=Image.Width;
 pImgAct->Graphique->Height=Image.Height;
 pImgAct->Graphique->Depth=Image.Depth;
 pImgAct->Graphique->ImageData=Image.ImageData;
 pImgAct->Graphique->PlanePick=Image.PlanePick;
 pImgAct->Graphique->PlaneOnOff=Image.PlaneOnOff;
 pImgAct->Graphique->NextImage=Image.NextImage;
 pImgAct->FenetreImage=NULL;
 pImgAct->OldGraphique=NULL;
 return(pImgAct);
}


void Detruire_Image(IMAGE_ACT *pImg)
{
 if(pImg->Graphique!=NULL)
 {
  int blargeur=((pImg->Graphique->Width+15)/16)*2;
  int len=pImg->Graphique->Depth*blargeur*pImg->Graphique-
>Height;
  FreeMem((char *) pImg->Graphique->ImageData, len);
  free(pImg->Graphique);
 }
 if(pImg->FenetreImage) Effacer_Image(pImg);
 free(pImg);
}


void Set_Image_FenetreImage(IMAGE_ACT *pImg, struct Window
*pFenetre)
{
 pImg->FenetreImage=pFenetre;
}

struct Window *Get_Image_FenetreImage(IMAGE_ACT *pImg)
{
 return(pImg->FenetreImage);
}


struct Window *Afficher_Image(IMAGE_ACT *pImg, struct Screen
*pEcran, USHORT posx, USHORT posy)
{
 struct Window *FImage;
 struct BitMap *supbitMap=NULL;
 ULONG rastersize;
 int i;
 SHORT W, H, D;


 struct NewWindow NFenImage =
 {
   0,0,
   0,0,
   0,1,
   MOUSEBUTTONS,
   REPORTMOUSE|NOCAREREFRESH|RMBTRAP|SUPER_BITMAP,
```

```
void Effacer_Image(IMAGE_ACT *pImg)
{
 int i;
 ULONG rastersize;
 struct Window *FImage;
 struct BitMap *supbitMap;

 FImage=Get_Image_FenetreImage(pImg);
 if(FImage!=NULL)
 {
  supbitMap=FImage->WLayer->SuperBitMap;
  rastersize=supbitMap->BytesPerRow * supbitMap->Rows;
  if(FImage)  CloseWindow(FImage);
  for(i=0;i<supbitMap->Depth;i++)
  {
   if(supbitMap->Planes[i])
   {
    FreeRaster(supbitMap->Planes[i],FImage->Width,FImage-
>Height);
    supbitMap->Planes[i]=NULL;
   }
  }
  FreeMem(supbitMap, sizeof(struct BitMap));
  supbitMap=NULL;
 }
 Set_Image_FenetreImage(pImg, NULL);
}


void MAJ_Image_Graphique(IMAGE_ACT *pImg)
{
 USHORT *pUSHORT;
 SHORT Width, Height, Depth;
 ULONG bitmaplen, rastersize;
 int blargeur,Mint,nbreUSHORT,i,j;
 struct Window *FImage;
 struct BitMap *supbitMap, *bitMap;

 FImage=Get_Image_FenetreImage(pImg);
 if(FImage!=NULL)
 {
  Mint=0xC0;
  Width=pImg->Graphique->Width;
  Height=pImg->Graphique->Height;
  Depth=pImg->Graphique->Depth;
  supbitMap=FImage->WLayer->SuperBitMap;
  bitMap=FImage->RPort->BitMap;
  blargeur=((Width+15)/16)*2;
  bitmaplen=(Height * blargeur * Depth);
  rastersize=supbitMap->BytesPerRow * supbitMap->Rows;
  nbreUSHORT=(int) ((rastersize+1)/2);
  BltBitMap(bitMap,(FImage->LeftEdge+2),(FImage-
>TopEdge+2),supbitMap,2,2,(FImage->Width-4),(FImage->Height-
4),Mint,0xFF,NULL);
  CopyMem(supbitMap->Planes[0], pImg->Graphique-
>ImageData,rastersize);
  pUSHORT=pImg->Graphique->ImageData;
  for(i=1;i<4;i++)
```

```
void Rotation_Droite_90_Image(struct BitMap *BMSource, struct
Image *Graphique, SHORT Width, SHORT Height, SHORT Depth, SHORT
TopEdge, SHORT LeftEdge)
{
 UBYTE coul;
 LONG result;
 ULONG rastersize, size;
 USHORT *pUSHORT;
 SHORT RWidth, RHeight, RDepth;
 int i, j, x, nbreUSHORT, Mint;
 struct RastPort *rastPort1;
 struct BitMap *BMResult=NULL;

 Mint=0xC0;



 RDepth=Depth;
 RWidth=(SHORT) Height;
 RHeight=(SHORT) Width;


 if(BMResult=(struct BitMap *) AllocMem(sizeof(struct BitMap),
MEMF_PUBLIC|MEMF_CLEAR))
 {
  InitBitMap (BMResult, RDepth, RWidth, RHeight);

  for(i=0;i<RDepth;i++)
  {
   BMResult->Planes[i]=(PLANEPTR) AllocRaster(RWidth, RHeight);
   if((BMResult->Planes[i])==NULL)
   {
    printf("Pas de place pour les BitMaps !!!");
    Exit(TRUE);
   }
   else BltClear(BMResult->Planes[i],RASSIZE(RWidth,
RHeight),0);
  }

  rastPort1=(struct RastPort *) malloc(sizeof(struct
RastPort));
  InitRastPort(rastPort1);
  rastPort1->BitMap=BMSource;

  for(j=0;j<Height-1;j++)
  {
   for(i=0;i<Width;i++)
    {
     x=RWidth-j;
     result=ReadPixel(rastPort1, LeftEdge+i, TopEdge+j);
     coul=(UBYTE) result;
     Ecrire_Pixel(BMResult, x, i, coul);
    }
  }

  free(rastPort1);

  rastersize=BMResult->BytesPerRow * BMResult->Rows;
```

```
    BMResult->Planes[i]=(PLANEPTR) AllocRaster(RWidth, RHeight);
    if((BMResult->Planes[i])==NULL)
    {
     printf("Pas de place pour les BitMaps !!!");
     Exit(TRUE);
    }
    else BltClear(BMResult->Planes[i],RASSIZE(RWidth,
RHeight),0);
   }

  rastPort1=(struct RastPort *) malloc(sizeof(struct
RastPort));
  InitRastPort(rastPort1);
  rastPort1->BitMap=BMSource;

  for(j=0;j<Height;j++)
  {
   for(i=0;i<Width;i++)
    {
     x=RHeight-(i+1);
     result=ReadPixel(rastPort1, LeftEdge+i, TopEdge+j);
     coul=(UBYTE) result;
     Ecrire_Pixel(BMResult, j, x, coul);
    }
  }

  free(rastPort1);


  rastersize=BMResult->BytesPerRow * BMResult->Rows;
  nbreUSHORT=(int) ((rastersize+1)/2);
  CopyMem(BMResult->Planes[0], Graphique->ImageData,
rastersize);
  pUSHORT=Graphique->ImageData;
  for(i=1;i<RDepth;i++)
  {
   for(j=1;j<=nbreUSHORT;j++)
   {
    pUSHORT++;
   }
   CopyMem(BMResult->Planes[i], pUSHORT, rastersize);
  }


  for(i=0;i<BMResult->Depth;i++)
  {
   if(BMResult->Planes[i])
   {
    FreeRaster(BMResult->Planes[i],RWidth,RHeight);
    BMResult->Planes[i]=NULL;
   }
  }
  FreeMem(BMResult, sizeof(struct BitMap));
  BMResult=NULL;

 }
}
```

```
 NouvFenetreMep.Screen=Ecran;
 NouvFenetreMep.Width=(SHORT) Misenpage->Dim.Largeur*taux;
 NouvFenetreMep.Height=(SHORT) Misenpage->Dim.Hauteur*taux;
 NouvFenetreMep.TopEdge=(SHORT) posy;
 NouvFenetreMep.LeftEdge=(SHORT) posx;

 if (!(FenetreMep = (struct Window *)
       OpenWindow(&NouvFenetreMep)))
       {
        printf("La fenetre Mise en Page ne peut s'ouvrir! \n");
        if(FenetreMep) CloseWindow(FenetreMep);
        exit(FALSE);
       }
 Misenpage->FenetreMep=FenetreMep;
 Case=Misenpage->Case;

 for(i=1;i<=Misenpage->NCases;i++)
 {
   Afficher_Case(FenetreMep, Misenpage->Dim, Case);
   Case=Case->NextCase;
 }
}

void Effacer_Mise_En_Page(MISE_EN_PAGE *pMep)
{
 if (pMep->FenetreMep) CloseWindow(pMep->FenetreMep);
 pMep->FenetreMep=NULL;
}

MISE_EN_PAGE *Init_Mise_En_Page(USHORT Largeur, USHORT Hauteur,
USHORT NCases, NEW_CASE TabNCases[])
{
 CASE *pCase1, *pCase2;
 MISE_EN_PAGE *pMep;
 int i;

 pMep = (struct MISE_EN_PAGE *) malloc(sizeof(MISE_EN_PAGE));
 pMep->Dim.Hauteur=Hauteur;
 pMep->Dim.Largeur=Largeur;
 pMep->NCases=NCases;

 if(0<NCases)
 {
  pCase1 =(CASE *) malloc(sizeof(CASE));
  pCase1->CoinSup=TabNCases[1].CoinSup;
  pCase1->CoinGauche=TabNCases[1].CoinGauche;
  pCase1->Dim.Hauteur=TabNCases[1].Dim.Hauteur;
  pCase1->Dim.Largeur=TabNCases[1].Dim.Largeur;
  pCase1->Numero=1;
  pCase1->EtatSelect=0;
  pCase1->NextCase=NULL;
  pMep->Case=pCase1;

  for(i=2;i<=NCases;i++)
  {
   pCase2 =(CASE *) malloc(sizeof(CASE));
   pCase1->NextCase=pCase2;
   pCase2->CoinSup=TabNCases[i].CoinSup;
```

```
  }
}

USHORT Get_Mise_En_Page_Nbre_Cases(MISE_EN_PAGE *pMep)
{
  return(pMep->NCases);
}
```

```
PRIVATE void Tracer_Rectangle(struct Window *Fenetre, RECTANGLE
DimPage, CASE *Case)
{
 SHORT fhaut, flarg;
 SHORT phaut, plarg;
 SHORT tablignes[10];


 fhaut = Fenetre->Height;
 flarg = Fenetre->Width;
 phaut = 296; /*DimPage.Hauteur;*/
 plarg = 210; /*DimPage.Largeur;*/


 tablignes[0]=Milli_2_Pix(Case->CoinGauche, flarg, plarg);
 tablignes[1]=Milli_2_Pix(Case->CoinSup, fhaut, phaut);
 tablignes[2]=tablignes[0]+Milli_2_Pix(Case->Dim.Largeur,
flarg, plarg);
 tablignes[3]=tablignes[1];
 tablignes[4]=tablignes[2];
 tablignes[5]=tablignes[1]+Milli_2_Pix(Case->Dim.Hauteur,
fhaut, phaut);
 tablignes[6]=tablignes[0];
 tablignes[7]=tablignes[5];
 tablignes[8]=tablignes[0];
 tablignes[9]=tablignes[1];

 SetDrPt(Fenetre->RPort, 0xFFFF);
 SetAPen(Fenetre->RPort, 1);
 Move(Fenetre->RPort, tablignes[0], tablignes[1]);
 PolyDraw(Fenetre->RPort, 5, &tablignes[0]);
}



DOC_PAGEBD *Init_PageBD(MISE_EN_PAGE *pMep, SHORT
NDessinsNNuls, NEW_DESSIN TabNDessins[])
{
 DESSIN *pDessin1, *pDessin2;
 NEW_DESSIN NouvDessin;
 DOC_PAGEBD *pPageBD;
 int i;

 pPageBD = (struct DOC_PAGEBD *) malloc(sizeof(DOC_PAGEBD));
 pPageBD->Mep=pMep;
 pPageBD->NDessins=pMep->NCases;
 pPageBD->NDessinsNNuls=NDessinsNNuls;
 pPageBD->Dessin=NULL;
 if(0<pPageBD->NDessins)
 {
  NouvDessin=TabNDessins[1];
  pDessin1=Init_Dessin(NouvDessin);
  pPageBD->Dessin=pDessin1;
  for(i=2; i<=pPageBD->NDessins; i++)
  {
    NouvDessin=TabNDessins[i];
    pDessin2=Init_Dessin(NouvDessin);
    pDessin1->NextDessin=pDessin2;
```

```
 NouvFenetrePage.Screen=Ecran;
 NouvFenetrePage.Width=(SHORT) pPageBD->Mep->Dim.Largeur*taux;
 NouvFenetrePage.Height=(SHORT) pPageBD->Mep->Dim.Hauteur*taux;
 NouvFenetrePage.TopEdge=(SHORT) posy;
 NouvFenetrePage.LeftEdge=(SHORT) posx;

 if (!(FenetrePage = (struct Window *)
       OpenWindow(&NouvFenetrePage)))
       {
        printf("La fenetre Page BD ne peut s'ouvrir! \n");
        if(FenetrePage) CloseWindow(FenetrePage);
        exit(FALSE);
       }

 pPageBD->FenetrePageBD=FenetrePage;
 pDessin=pPageBD->Dessin;
 pCase=pPageBD->Mep->Case;

 for(i=1;i<=pPageBD->NDessins;i++)
 {
   if(pDessin->Graphique->ImageData==NULL)
   {
    Tracer_Rectangle(FenetrePage, pPageBD->Mep->Dim, pCase);
   }
   else
   {
    top=Milli_2_Pix(pCase->CoinGauche, FenetrePage->Width,
210);
    left=Milli_2_Pix(pCase->CoinSup, FenetrePage->Height, 296);
    larg=Milli_2_Pix(pCase->Dim.Largeur, FenetrePage->Width,
210);
    haut=Milli_2_Pix(pCase->Dim.Hauteur, FenetrePage->Height,
296);
    ntaille.Largeur=larg;
    ntaille.Hauteur=haut;

    Graphique=Reduire_Dessin(pDessin, ntaille);
    DrawImage(FenetrePage->RPort, Graphique, top, left);
    blargeur=((Graphique->Width+15)/16)*2;
    bitmaplen=Graphique->Height*blargeur*Graphique->Depth;
    FreeMem(Graphique->ImageData,bitmaplen);
    FreeMem(Graphique, sizeof(struct Image));
    Tracer_Rectangle(FenetrePage, pPageBD->Mep->Dim, pCase);
   }
   pDessin=pDessin->NextDessin;
   pCase=pCase->NextCase;
 }
}


void Effacer_PageBD(DOC_PAGEBD *pPageBD)
{
 if (pPageBD->FenetrePageBD) CloseWindow(pPageBD-
>FenetrePageBD);
 pPageBD->FenetrePageBD=NULL;
}
```

```
/*
 * ---> Objet     : Contexte de réalisation
 * ---> Type      : Source complète
 * ---> Auteurs   : Tillieux Marc
                    Pequet Benoît
*/


#include <stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<exec/types.h>
#include<intuition/intuition.h>
#include "exec/memory.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "h_privacy.h"
#include "h_ObjetsBD.h"
#include "h_Case.h"
#include "h_PageBD.h"
#include "h_Image.h"


PRIVATE CONTEXT
Contexte={NULL,NULL,NULL,NULL,0,0,0,NULL,NULL,NULL,NULL,NULL,NU
LL};
PRIVATE NEW_MEP NMep[4];
PRIVATE NEW_CASE NCase[4][10];


void Get_Contexte_New_Mep(int i, NEW_MEP *NouvMep, NEW_CASE
NouvCase[]);
int Get_Contexte_Nbre_New_Mep(void);
void Get_Contexte_New_Dessins(MISE_EN_PAGE *pMep, NEW_DESSIN
NouvDessin[]);
void Set_Contexte_Mep(MISE_EN_PAGE *pMep);
MISE_EN_PAGE *Get_Contexte_Mep(void);
DOC_PAGEBD *Get_Contexte_PageBD(void);
void Selectionne_Contexte_Case(DOC_PAGEBD *pPageBD, int
NumCase);
CASE *Get_Contexte_Case_Selectionnee(void);
DESSIN *Get_Contexte_Dessin_Selectionne(void);
void Set_Contexte_PageBD(DOC_PAGEBD *pPageBD);
void Reinit_Contexte(void);
UBYTE Get_Contexte_Couleur_Fond(void);
void Set_Contexte_Couleur_Fond(UBYTE Couleur);
UBYTE Get_Contexte_Couleur_Actuelle(void);
void Set_Contexte_Couleur_Actuelle(UBYTE Couleur);
void Set_Contexte_Pointeur_Image_Dessin(USHORT
*pImageDataDessin);
void Save_Contexte_PFast(USHORT *pFastRam);
USHORT *Get_Contexte_PFast(void);
char *Get_Contexte_BISec(void);
char *Get_Contexte_ZoneTrav(void);
char *Get_Contexte_Nom_Util(void);
```

```
NCase[2][1].Dim.Largeur = 50;
NCase[2][1].Dim.Hauteur = 80;

NCase[2][2].CoinSup = 20;
NCase[2][2].CoinGauche = 75;
NCase[2][2].Dim.Largeur = 120;
NCase[2][2].Dim.Hauteur = 80;

NCase[2][3].CoinSup = 110;
NCase[2][3].CoinGauche = 15;
NCase[2][3].Dim.Largeur = 80;
NCase[2][3].Dim.Hauteur = 80;

NCase[2][4].CoinSup = 110;
NCase[2][4].CoinGauche = 105;
NCase[2][4].Dim.Largeur = 90;
NCase[2][4].Dim.Hauteur = 80;

NCase[2][5].CoinSup = 200;
NCase[2][5].CoinGauche = 15;
NCase[2][5].Dim.Largeur = 120;
NCase[2][5].Dim.Hauteur = 80;

NCase[2][6].CoinSup = 200;
NCase[2][6].CoinGauche = 145;
NCase[2][6].Dim.Largeur = 50;
NCase[2][6].Dim.Hauteur = 80;



NMep[3].Dim.Hauteur = 296;
NMep[3].Dim.Largeur = 210;
NMep[3].NCases = 5;
NMep[3].TabNCase = NULL;

NCase[3][1].CoinSup = 20;
NCase[3][1].CoinGauche = 15;
NCase[3][1].Dim.Largeur = 130;
NCase[3][1].Dim.Hauteur = 80;

NCase[3][2].CoinSup = 20;
NCase[3][2].CoinGauche = 155;
NCase[3][2].Dim.Largeur = 40;
NCase[3][2].Dim.Hauteur = 80;

NCase[3][3].CoinSup = 110;
NCase[3][3].CoinGauche = 15;
NCase[3][3].Dim.Largeur = 40;
NCase[3][3].Dim.Hauteur = 80;

NCase[3][4].CoinSup = 110;
NCase[3][4].CoinGauche = 65;
NCase[3][4].Dim.Largeur = 130;
NCase[3][4].Dim.Hauteur = 80;

NCase[3][5].CoinSup = 200;
NCase[3][5].CoinGauche = 15;
NCase[3][5].Dim.Largeur = 180;
```

```
void Selectionne_Contexte_Case(DOC_PAGEBD *pPageBD, int
NumCase)
{
   MISE_EN_PAGE *pMep;
   CASE *pCase1, *pCase2;


   pMep=pPageBD->Mep;

   if((NumCase<=pMep->NCases) && (NumCase>0))
   {
    pCase1=pMep->Case;

    while(pCase1!=NULL)
    {
     pCase2=pCase1->NextCase;
     if(pCase1->Numero==NumCase)
     {
      Selectionner_Case(pCase1);
      Contexte.CaseSel=pCase1;
      Contexte.DessinSel=Get_PageBD_Dessin(NumCase, pPageBD);
     }
     else
     {
      Deselectionner_Case(pCase1);
     }
     pCase1=pCase2;
    }
   }
}


CASE *Get_Contexte_Case_Selectionnee(void)
{
  return(Contexte.CaseSel);
}


DESSIN *Get_Contexte_Dessin_Selectionne(void)
{
  return(Contexte.DessinSel);
}


void Set_Contexte_PageBD(DOC_PAGEBD *pPageBD)
{
   Contexte.pointPageBD=pPageBD;
   Set_Contexte_Mep(pPageBD->Mep);
   Selectionne_Contexte_Case(Contexte.pointPageBD, 1);
}


void Reinit_Contexte(void)
{
  Contexte.NomUtil="Marc\0";
  Contexte.Drive="WORK:";
  Contexte.ZoneTrav="LC/MARC/ZTRAV";
  Contexte.BISec="LC/MARC/ZTRAV/BISEC";
```

```
char *Get_Contexte_ZoneTrav(void)
{
 return(Contexte.ZoneTrav);
}


char *Get_Contexte_Nom_Util(void)
{
 return(Contexte.NomUtil);
}


struct IMAGE_ACT *Get_Contexte_Image_Active(void)
{
 if(Contexte.ImageActive->Graphique!=NULL)
 {
  return(Contexte.ImageActive);
 }
 else return(NULL);
}


void Detruire_Contexte_Image_Active(void)
{
 Detruire_Image(Contexte.ImageActive);
 Contexte.ImageActive=NULL;
}


void Set_Contexte_Image_Active(IMAGE_ACT *im)
{
 Contexte.ImageActive=im;
}
```

```
 for(count1=0;count1<hauteur;count1++)
  for(count2=0;count2<profondeur;count2++)
   {
    posit=count2*bplan+(count1*blargeur);
    compteur=0;
    while(compteur<blargeur)
     {
      fgetst(&code,1,fd);
      if(code<128)
       {
        fgetst(&bitmap[posit+compteur],code+1,fd);
        compteur=compteur+code+1;
       }
      else
       if(code>128)
        {
         fgetst(&buffer,1,fd);
         for(count3=compteur;count3<(compteur+257-
code);count3++)
          bitmap[posit+count3]=buffer;
         compteur=compteur+257-code;
        }
       else;
      }
   }
}

PRIVATE void LectBitmap(fd, bitmap, hauteur, largeur,
profondeur)
FILE *fd;
UBYTE *bitmap, profondeur;
USHORT hauteur, largeur;
{
 int blargeur=((largeur+15)/16)*2, bplan=hauteur*blargeur;
 int count1, count2, posit;
 for(count1=0;count1<hauteur;count1++)
  for(count2=0;count2<profondeur;count2++)
   {
     posit=count2*bplan+(count1*blargeur);
     fgetst(&bitmap[posit],blargeur,fd);
    }
}



PRIVATE void EcritBitmap(fd, bitmap, hauteur, largeur,
profondeur)
FILE *fd;
UBYTE *bitmap;
USHORT profondeur, hauteur, largeur;
{
 int blargeur=((largeur+15)/16)*2, bplan=hauteur*blargeur;
 int count1, count2, posit;
 for(count1=0;count1<hauteur;count1++)
  for(count2=0;count2<profondeur;count2++)
   {
     posit=count2*bplan+(count1*blargeur);
     fwrite(&bitmap[posit],blargeur,1,fd);
```

```
        else if(identificateur==CRNGID)
         {
          for(count=0;count<4;count++)
          {
           fgetst((char *) &imagedata->crng[count],len,fd);
           crngcount++;
          }
         }
        else if(identificateur==CAMGID)
         {
          fgetst((char *) imagedata->camg,len,fd);
         }
        else if(identificateur==BODYID)
         {
          imagedata->body=NULL;
          blargeur=((imagedata->bmhd.w+15)/16)*2;
          bitmaplen=imagedata->bmhd.h*blargeur*(imagedata-
>bmhd.nPlanes);
          if((imagedata->body=(USHORT *)
           AllocMem(bitmaplen,MEMF_CHIP | MEMF_CLEAR))==NULL)
            {
             printf("ERREUR manque de CHIP RAM\n");
             free(imagedata->cmap);
             exit(1);
            }
          if(imagedata->bmhd.compression==0)
           LectBitmap(fd,imagedata->body,imagedata->bmhd.h,
                      imagedata->bmhd.w,imagedata-
>bmhd.nPlanes);
          else
           Decompress(fd,imagedata->body,imagedata->bmhd.h,
                      imagedata->bmhd.w,imagedata-
>bmhd.nPlanes);
         }
       }
     }
   }
 fclose(fd);
}



boolean
ChargerImage(nom,picto,palette,viewmode,scwidth,scheight)
char *nom;
struct Image *picto;
USHORT *palette;
long *viewmode;
WORD scwidth, scheight;
{
 boolean iffilbm=TRUE;
 ILBM_16 imagedata;
 int i,len1,result=0;
 UBYTE *couleur;
 USHORT *couleur1;
 result = ChargeIlbm(nom,&imagedata);
 if (result!=1)
 {
```

```
  i++;
 }
 while(error==0)
 {
  error = dnext(info);
  if(error==0)
  {
   stpcpy(tabnom[i],stpblk(info->fib_FileName));
   i++;
  }
 }
 free(info);
 return(i);
}


int Get_Disque_Fichiers(char *nom, char tabnom[][20])
{
 int i;

 i=Get_Disque_Info(nom, 0, tabnom);
 return(i);
}


int Get_Disque_Directories(char *nom, char tabnom[][20])
{
 int i;

 i=Get_Disque_Info(nom, 1, tabnom);
 return(i);
}


boolean Appartient_Disque(char *chemin, char *nom)
{
 boolean res=FALSE;
 int i, max, x;
 char tab[30][20];

 max=Get_Disque_Fichiers(chemin, tab);

 for(i=0;i<max;i++)
 {
  x=strcmpi(nom,tab[i]);
  if(x==0) {res=TRUE;}
 }
 return(res);
}


void Sauver_Disque_MEP(MISE_EN_PAGE *pMep, char *drive, char
*chemin, char *nom)
{
 int i;
```

```
  fwritest((char *) &pDessin->NumCase, 2, fd);
  fwritest((char *) &pDessin->Graphique->Height, 2, fd);
  fwritest((char *) &pDessin->Graphique->Width, 2, fd);
  fwritest((char *) &pDessin->Graphique->Depth, 2, fd);
  EcritBitmap(fd, pDessin->Graphique->ImageData, pDessin-
>Graphique->Height, pDessin->Graphique->Width, pDessin-
>Graphique->Depth);
  pDessin=pDessin->NextDessin;
 }

 fclose(fd);
}



int Charger_Disque_MEP(char *drive, char *chemin, char *nom,
USHORT *L, USHORT *H, USHORT *NCases, NEW_CASE TabNCases[])
{
 int i;
 FILE *fd;
 char  nomMep[FMSIZE], *MEPID=NULL, *id=NULL;


 MEPID=stpcpy(MEPID,"MEP\0");

 strmfn(nomMep, drive, chemin, nom, "MEP");
 fd=fopen(nomMep,"r");


 fgetst((char *) id, 3, fd);
 if(*id!=*MEPID)
 {
  fclose(fd);
  return 1;
 }
 else
 {
  fgetst((char *) H, 2, fd);
  fgetst((char *) L, 2, fd);
  fgetst((char *) NCases, 2, fd);
  for(i=1; i<=*NCases; i++)
  {
   fgetst((char *) &TabNCases[i].CoinSup, 2, fd);
   fgetst((char *) &TabNCases[i].CoinGauche, 2, fd);
   fgetst((char *) &TabNCases[i].Dim.Hauteur, 2, fd);
   fgetst((char *) &TabNCases[i].Dim.Largeur, 2, fd);
  }
 }
 fclose(fd);
}



int Charger_Disque_PageBD(char *drive, char *chemin, char *nom,
USHORT *L, USHORT *H, USHORT *NCases, NEW_CASE TabNCases[],
USHORT *DNNuls, NEW_DESSIN TabNDessins[])
{
 int i, blargeur, bitmaplen;
```

```
    Graphique->TopEdge=0;
    Graphique->Width=Largeur;
    Graphique->Height=Hauteur;
    Graphique->Depth=Profondeur;
    Graphique->PlanePick=31;
    Graphique->PlaneOnOff=31;
    Graphique->NextImage=NULL;
    LectBitmap(fd,(UBYTE *) Graphique->ImageData, Hauteur,
Largeur, Profondeur);

    TabNDessins[NumCase].Graphique=Graphique;
    Graphique=NULL;
   }
  }
 fclose(fd);
}
```

```
PRIVATE void DeletePrtReq(request)
union printerIO *request;
{
 struct MsgPort *prtport;
 prtport=request->ios.io_Message.mn_ReplyPort;
 DeleteExtIO((struct IORequest *) request);
 DeletePort(prtport);
}


PRIVATE int OpenPrinter(request)
union printerIO *request;
{
 return(OpenDevice("printer.device",0,(struct IOrequest
*)request,0));
}


PRIVATE void DumpRPort(request, rastport, colormap, modes, sx
,sy, sw, sh, dc, dr, s)
union printerIO *request;
struct RastPort *rastport;
struct ColorMap *colormap;
ULONG modes;
UWORD sx, sy, sw, sh;
LONG dc, dr;
UWORD s;
{
 request->iodrp.io_Command = PRD_DUMPRPORT;
 request->iodrp.io_RastPort = rastport;
 request->iodrp.io_ColorMap = colormap;
 request->iodrp.io_Modes = modes;
 request->iodrp.io_SrcX = sx;
 request->iodrp.io_SrcY = sy;
 request->iodrp.io_SrcWidth = sw;
 request->iodrp.io_SrcHeight = sh;
 request->iodrp.io_DestCols = dc;
 request->iodrp.io_DestRows = dr;
 request->iodrp.io_Special = s;
 SendIO((struct IORequest *) request);
}


int print(rport, Fenetre, posx, posy, width, height)
struct RastPort *rport;
struct Window *Fenetre;
short posx, posy, width, height;
{
 struct IntuiMessage *msg;
 struct MsgPort *port;
 struct RastPort *rp;
 struct ViewPort *vp;
 ULONG usersig, printersig, signal;
 boolean fin=False;

 if(!(printerReq = CreatePrtReq()))
 {
  printf("Erreur de cr0ation de l'imprimante\n");
```

Réalisation, impression et reliure : SPRL Centre de Reprographie - Tél. : 081/22.87.72